INTERNET EXPLORER VA IN TILT
HACKING: SOTTO LA LENTE TRE INEDITI EXPLOIT

VERSIONE PLUS ✔ RIVISTA+LIBRO+CD €14,90
VERSIONE STANDARD ☐ RIVISTA+CD €6,90

ioPROGRAMMO

Poste Italiane • Spedizione in a.p. - 45% • art. 2 comma 20/b legge 662/96 - AUT. N. DCDC/033/01/CS/CAL Periodicità mensile • LUGLIO/AGOSTO 2004 • ANNO VIII, N.7 (82)

IL WEB DIVENTA MULTIMEDIALE

- ✔ Sfruttare XML per riprodurre dinamicamente MP3
- ✔ Nuove tecniche di streaming con Flash MX
- ✔ Il codice javascript per "far parlare" il tuo sito

SUL CD Il progetto completo

IN ALLEGATO

SISTEMA

- Smart Document: fatture automatiche con XML e Office
- Agent: come animare gli assistenti in stile Office
- Creare un RDBMS in Java
- Crittografia facile! Basta un ActiveX
- VB: controllare l'hardware di un PC remoto

MULTIMEDIA

- Muovere gli oggetti nello spazio 3D con .NET
- Gestire i file nelle applicazioni di grafica

CORSI

- Java: scrivi meno codice con il polimorfismo
- C#: delegati ed eventi per applicazioni estensibili
- C++: i ritocchi per ottimizzare il codice

ADVANCED

- Programmazione Aspect Oriented con AspectJ
- La simulazione di fenomeni reali in 3D

SVILUPPARE APPLICAZIONI PER TELEFONINI J2ME

Dalla A alla Z: tutti gli strumenti ed il codice sorgente per creare la tua applicazione MOBILE

CODICE A BARRE ELETTRONICO

Identificare a distanza i prodotti di un magazzino

COMANDARE UN ROBOT DAL PC

Come costruire un robot semovente con meno di 20 €

ISSN 1128-594X
9 771128 594641 40082
ioProgrammo Plus Anno VIII - N° 7 (82) • € 14,90
EDIZIONI MASTER
www.edmaster.it

VB.NET: REALIZZARE APPLICAZIONI DI GRAFICA CON GDI+

Anno VIII - n. 7 (82) Luglio-Agosto 2004

Strade

Questo mese ospitiamo le ultime puntate di ben due corsi: quello storico di C++ ed il più giovane, ma altrettanto valido, su Delphi. Non vi anticipo le novità con cui saranno sostituiti, vi dico solo che cercheremo di venire in contro alle richieste che da molti lettori sono giunte in merito al livello di difficoltà degli articoli. I passi che abbiamo in mente di compiere vogliono arrivare a snellire e rendere più "leggibili" gli articoli che finora sono sembrati più difficili da comprendere.
Il nostro impegno è sempre stato quello di accelerare la curva di apprendimento delle tecnologie che proponiamo, attraverso la sinergia fra una trattazione il più possibile chiara ed esempi applicativi che abbiano una reale impatto in ambito lavorativo o hobbistico. Ovviamente, non abbandoneremo questa strada: anche nel titolo di copertina di questo mese emergono le due anime di ioProgrammo. Da un lato la parte più professionale che guarda all'accessibilità dei siti, dall'altra la voglia di giocare con le tecnologie e meravigliare noi stessi: dare la voce ad un sito Web, spiegando sia come farlo cambiando semplicemente i file audio da riprodurre, sia la tecnologia lato server che può interessare ai più esperti. In realtà, questa dicotomia fra professionale e divertimento non ci è mai andata giù. Il nostro scopo è lavorare con entusiasmo e suscitare il vostro piacere di programmare. Le vostre mail e l'incredibile partecipazione al sito di ioProgrammo ci confermano nelle nostre scelte. Comunque, per qualsiasi correzione di rotta: attendo le vostre critiche!

rdelmonaco@edmaster.it

All'inizio di ogni articolo, troverete un nuovo simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.
Per scaricare software e codice da Internet, ogni mese indicheremo una password differente. Per il numero che avete fra le mani la combinazione è:

Username: **fibonacci** *Password:* **112358**

ioProgrammo

Anno VIII - N.ro 7 (82) - Luglio/Agosto 2004 - Periodicità: Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale Massimo Sesti
Direttore Responsabile Romina Sesti
Responsabile Editoriale Gianmarco Bruni
Responsabile Marketing Antonio Meduri
Editor Gianfranco Forlino
Coordinamento redazionale Raffaele del Monaco
Redazione Antonio Pasqua, Thomas Zaffino
Collaboratori R. Allegra, M. Autiero, L. Barbieri, M. Battista, M. Bigatti, L. Buono, F. Diotalevi, M. Del Gobbo, E. Florio, F. Grimaldi, A. Ingegneri, F. Lippo, M. Locuratolo, A. Marroccelli, F. Mestrone, C. Pelliccia, P. Perrotta, S. Russo, L. Spuntoni, F. Vaccaro, C. F. Zoffoli.
Segreteria di Redazione Veronica Longo

Realizzazione grafica Cromatika S.r.l.
Responsabile grafico: Paolo Cristiano
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri, R. Del Bo
Impaginazione elettronica: Aurelio Monaco

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

Certificato UNI EN ISO 14001
N. 9191 CRMT

Realizzazione Multimediale SET S.r.l.
Coordinamento Tecnico Piero Mannelli
Realizzazione CD-Rom Paolo Iacona
Pubblicità Master Advertising s.r.l.
Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite Daisy Zonato

Editore Edizioni Master S.r.l.
Sede di Milano: Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121206
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Amministratore Unico: Massimo Sesti

Abbonamento e arretrati
ITALIA: Abbonamento Annuale: ioProgrammo Basic (11 numeri) + mini hub USB: €52,90 sconto 30% sul prezzo di copertina di €75.90
ioProgrammo Libro (11 numeri + libro) €86,90 sconto 30% sul prezzo di copertina di €123.90
Offerte valide fino al 30/09/04
ESTERO: Abbonamento Annuale: ioProgrammo Basic (11 numeri): € 151,80. ioProgrammo Plus (11 numeri + 6 libri): € 257,00
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via Cesare Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- **cc/p n.16821878** o **vaglia postale** (inviando copia della ricevuta del versamento insieme alla richiesta);
- **assegno bancario non trasferibile** (da inviarsi in busta chiusa insieme alla richiesta);
- **carta di credito**, circuito VISA, CARTASI', MASTERCARD/EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).
- **bonifico bancario** intestato a Edizioni Master S.r.l. c/o Banca Credem S.p.a. c/c 01 000 000 5000 ABI 03032 CAB 80880 CIN Q (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.
Sostituzioni: Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master Servizio Clienti - Via Cesari Correnti, 1 - 20123 Milano

Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
tel.02 831212
@ e-mail: servizioabbonati@edmaster.it

Stampa: Rotoeffe Via Variante di Cancelliera, 2/6 - Ariccia (Roma)
Stampa CD-Rom: Neotek S.r.l.- C.da Imperatore- zona ASI - Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Giugno 2004

Certificato n.5113 del 25/11/2003

Edizioni Master edita:
Idea Web, Go!OnLine Internet Magazine, Win Magazine, PC Fun extreme, Quale Computer, DVD Magazine, Office Magazine, La mia Barca, ioProgrammo, Linux Magazine, Softline Software World, HC Guida all'Home Cinema, MPC, Discovery DVD, Computer Games Gold, inDVD, I Fantastici CD-Rom, PC VideoGuide, I Corsi di Win Magazine, I Filmissimi in DVD, La mia videoteca, Tv & Satellite, Win Extra, Home entainment, Digital Japan, ioProgrammo Extra, Le Collection.

MICROSOFT E ORACLE ASSIEME PER LO SVILUPPO

I due giganti hanno siglato un accordo che prevede l'integrazione fra Visual Studio Net 2003 ed il database Oracle.
Dopo anni di feroce battaglia, l'accordo stretto fra i due giganti consentirà agli sviluppatori di scrivere applicazioni per DB Oracle, utilizzando Visual Studio .Net 2003.
Sembra dunque proseguire la strategia che vede Microsoft sistemare pacificamente i conti con tutte le aziende che l'avevano veementemente attaccata in passato.

www.oracle.com

I DATABASE RIPRENDONO A CRESCERE

Le vendite di database sono sempre state prese a misura dello stato generale dell'industria informatica. Nel 2002, la crisi del settore si era riflessa in un -6% delle vendite di DB, mentre il 2003 ha visto una crescita complessiva del 5%.
Sugli scudi troviamo i soliti nomi: IBM, Oracle e Microsoft. DB2 di IBM guida la classifica con una quota del 36%, mentre Oracle e Microsoft si fermano, rispettivamente, al 32,6% e al 18,7%. Il database costituisce l'infrastruttura software che precede la costruzione di qualsiasi nuovo progetto. Ecco perché e così importante valutare la dinamica di questo mercato. Il crollo delle vendite dei DB nel 2002 fu infatti diretta conseguenza del disimpegno di molte aziende in nuovi progetti mentre oggi la prospettiva è completamente diversa.
Indicativo è lo straordinario aumento di database basati su Linux +100% nel 2003, aumento che porta 300 milioni di dollari il mercato complessivo dei DB Linux.

JOBS E TIGER SARANNO LE STELLE DELLA WWDC 2004

Sarà Steve Jobs ad aprire l'edizione 2004 della Worldwide Developers Conference, con un'anteprima di Mac OS X "Tiger" La keynote di quest'anno includerà un'anteprima di "Tiger". L'evento, della durata di cinque giorni, avrà inizio il 28 giugno e si concluderà il 2 luglio, ospiterà quasi 200 sessioni tecniche con nuovi contenuti studiati per soddisfare un'ampia gamma di sviluppatori Mac.

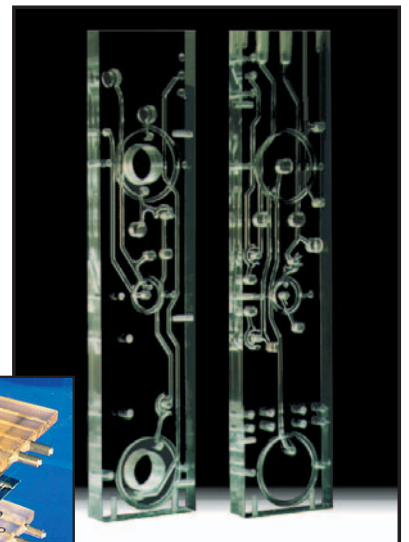
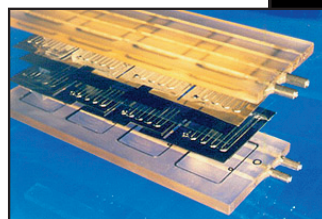
Uno sguardo in profondità alle ultime tecnologie Mac OS X, laboratori "hands-on" con gli ultimi sistemi Mac, sezioni più estese per l'Enterprise IT e una sezione dedicata per gli sviluppatori QuickTime e i creatori di contenuti. I partecipanti potranno toccare con mano le tecnologie di Mac OS X, sco-

News

CRACCATA LA CHIAVE CRITTOGRAFICA DA 576 BIT

Era la scommessa rivolta ai ricercatori da RSA Security: il crack della chiave crittografica lo si deve ad un team di matematici americani che, utilizzando un centinaio di workstation e poco più di tre mesi di elaborazione, sono riusciti nell'impresa.
"Le informazioni ricevute durante questi concorsi costituiscono una risorsa di grande valore sia per chi studia la crittografia, sia per chi deve scegliere il giusto livello di sicurezza", ha affermato Burt Kaliski, chief scientist and director di RSA Laboratories.

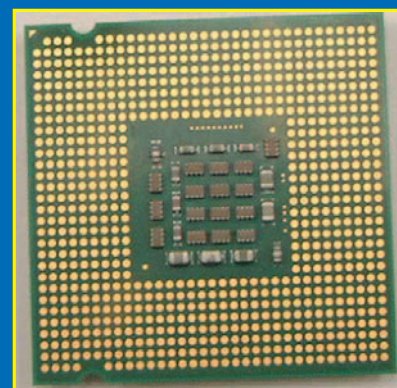
CELLE A COMBUSTIBILE GIÀ NELL'ANNO 2005

La ABI Research ha annunciato che cellulari, laptop e palmari, potrebbero essere alimentati da celle a combustibile già dal 2005. Le celle a combustibile (micro fuel cells o MFC) promettono una durata molto maggiore delle attuali batterie, riducendo al contempo i tempi di rcarica. La ricerca ha evidenziato anche le incertezze di questa nuova tecnologia, incertezze che si rifletteranno in una lenta penetrazione del mercato: nel 2012, non più del 15 percento dei PC portatili saranno equipaggiati con MFC.

prendone le fondamenta open source le nuove applicazioni, e i tool di sviluppo di prossima generazione. Gli sviluppatori aziendali, gli amministratori di sistemi e gli IT manager potranno partecipare ad una sessione Enterprise IT più estesa, studiata specificamente per aiutare a capire le tecnologie chiave che saranno fornite in Tiger e sfruttarle appieno nei loro ambienti. QuickTime e i Digital Media avranno acnh'essi una sessione per sviluppatori e creatori di contenuti che vogliono esplorare le ultime tecnologie in ambito multimedia: distribuzione dei contenuti con QuickTime Streaming Server, e nuovi tool e tecniche per la creazione e la distribuzione di contenuti attraverso media digitali. Ovviamente, non mancheranno laboratori migliorati "hands-on" con gli ultimi sistemi Mac, dove gli sviluppatori potranno testare il loro codice e ricevere assistenza tecnica dagli ingegneri Apple.

http://developer.apple.com/wwdc/

INTEL SBATTE CONTRO UN MURO TERMICO

Il più grande produttore di chip al mondo ha annunciato una svolta nello sviluppo dei futuri processori: due progetti (nome in codice Tejas e Jayhawk) saranno abbandonati a causa di un grave problema di fisica. In pratica, una sorta di muro termico impedisce l'aumento della velocità di clock oltre una certa soglia.
Proprio sull'aumento della velocità puntavano invece i due progetti abbandonati al loro posto, la nuova strategia di Intel prevede un più spinto parallelismo: si tenderà a integrare in un unico chip sempre più processori, in modo da bilanciare le difficoltà nell'aumentare la velocità di clock.

www.intel.com

BEA, IL FUTURO È LIQUIDO

Il tema della eWorld conference di quest'anno è la Service Oriented Architecture (SOA). Nella keynote di apertura Alfred Chuang, presidente di BEA, ha lanciato il messaggio sottolineato l'importanza di cominciare, da subito, a sviluppare in ambito SOA. Gli ultimi sforzi di BEA sono stati tutti volti a rendere WebLogic più facile da interfacciare con applicazioni prodotte da terze parti. Sempre secondo Chuang, l'obiettivo comune di BEA e degli sviluppatori sarà quello di raggiungere il "Liquid Computing": sistemi flessibili al punto da adattarsi da soli a qualsiasi ostacolo, aggirandolo. Come fossero fluidi, appunto.
Il continuo aggiornamento cui è sottoposto ormai ogni sistema software deve dunque in qualche modo "automatizzarsi": sistemi adattivi dovranno consentire l'adeguamento al mutamento delle esigenze e dell'ambiente a costo zero. In questa ottica, l'architettura Services Oriented è solo il primo passo verso il reale obiettivo: il liquid computing.

www.bea.com

MICROSOFT TECHED 2004 EUROPE

Dal 29 giugno al 2 luglio 2004, Amsterdam sarà invasa da migliaia di sviluppatori, la città sarà infatti teatro del principale appuntamento europeo di Microsoft: il Teched 2004. Per sviluppatori ed i professionisti dell'IT sarà l'occasione per approfondire temi legati allo sviluppo applicativo, al deployment, alla sicurezza e alla gestione di soluzioni pensate per soddisfare le esigenze di business.
Si avrà modo di vedere in azione sia tecnologie e prodotti già rilasciati sia quelli disponibili nell'arco dei sei mesi successivi all'evento. La conferenza consiste di 4 giornate precedute da una pre-conference il 28 giugno 2004. Come sempre, ioProgrammo coprirà l'evento con reportage e interviste in esclusiva.

www.microsoft.com

MOTORI DI RICERCA LA SFIDA DEL MULTIMEDIA

I contenuti testuali sono stati, da sempre, la colonna portante di Internet. La generazione di Internet ha vissuto un vero e proprio ritorno alla parola stampata: dopo anni in cui televisione e cinema avevano quasi del tutto soppiantato la comunicazione scritta, il testo è tornato prepotentemente alla ribalta. Ma, come diciamo da molto tempo, questa condizione non è destinata a durare ancora molto: la larghezza di banda consente (e consentirà sempre più) la diffusione di contenuti multimediali (essenzialmente audio e filmati) che, oltre a cambiare la natura "sociologica" di Internet, ne modificheranno dal profondo alcune caratteristiche eminentemente tecniche. Una su tutte: le ricerche. faranno i motori di ricerca a estrapolare ed archiviare la massa di informazioni audiovisiva che si sta riversando nell'universo ▸▸

RILASCIATO FILEMAKER SERVER 7

Grandi performance per il nuovo software server di FileMaker, che può ospitare una quantità di dati illimitata offrendo la massima facilità nella gestione dei database condivisi. FileMaker Server 7 consente di gestire e condividere i database creati con FileMakerPro 7 ed è finalmente disponibile sul mercato italiano. Ad annunciarlo, FileMaker Inc., azienda californiana leader mondiale nella produzione di software database per workgroup aziendali e singoli utenti. Ryan Rosemberg, Vice Presidente della divisione Marketing and Service di FileMaker ha dichiarato: "FileMaker Server 7 permette alla piccolissima azienda così come alla grande realtà imprenditoriale con più gruppi di lavoro di condividere, gestire e accedere alle informazioni contenute nel database in modo semplice offrendo la massima produttività". FileMaker Server 7 assicura numerosi potenziamenti tecnologici per condividere i dati e per la loro amministrazione, oltre ad offrire funzioni avanzate in grado di proteggere le informazioni. FileMaker Server 7 ha una nuova funzione che permette di eseguire le ricerche e i calcoli direttamente sul server, invece che sul client; per questo é in grado di gestire i database in modo più rapido, ottimizzando, inoltre, i sistemi di memorizzazione dell'hard disk e dei server multi-CPU. Un altro vantaggio offerto dalla nuova versione é la grande quantità di RAM sempre disponibile, procedimento ottenuto mediante i nuovi e sofisticati sistemi di caching.

FileMaker Server 7 può ospitare sino a 125 file di database e ogni database è in grado di memorizzare fino a 8 terabyte di informazioni, una capacità 4.000 volte superiore al precedente limite.

Ogni file del database può ora contenere tabelle multiple di dati, e un singolo FileMaker Server puòospitare milioni di tabelle. Per espanderne ulteriormente la capacità, l'utente può aggiungere ulteriori FileMaker Server al proprio network a seconda delle esigenze di crescita del business.

www2.filemaker.fr/italy/products/fms_home.html

Internet? Attualmente, la catalogazione delle informazione multimediali si poggia semplicemente sulla descrizione testuale che accompagna i filmati e i file audio presenti sul Web. Anche la ricerca per immagini presente in Google e in altri motori di ricerca si basa sul nome del file o sulle descrizioni che si trovano nelle vicinanze dell'immagine nella pagina Web originale. La catalogazione non è dunque effettuata direttamente sull'immagine, o sui contributi audio-video, ma sul testo che li accompagna. Tutti i principali motori di ricerca, Google in testa, sono impegnati nello sviluppo di tecnologie che consentano di estrapolare dati utili alle ricerche direttamente dalle fonti audio-video. In questa corsa, una piccola compagnia, la SreamSage, sembra aver accumulato un discreto vantaggio: la tecnologia che ha sviluppato si occupa di trascrivere il parlato presente nei contributi audio, consentendo la ricerca sulla reale fonte. I risultati della ricerca non solo indicano in quali file è presente il testo cercato ma evidenziano anche il punto del file in cui è citata la chiave di ricerca. In via sperimentale, potete provare il servizio su *www.campaign-search.com*, in cui sono stati catalogati i discorsi dei due candidati alle prossime elezioni americane.

www.streamsage.com

CA RIVELA IL CODICE DI INGRES

La Computer Associates ha annunciato il rilascio del codice del suo popolare database Ingres. Sede dell'annuncio è stato il CA World del maggio scorso: chiaro segno dell'attenzione che CA rivolge al movimento Open Source e a Linux. Ad accompagnare il rilascio del codice, è stata messa a punto nuova licenza: CA Trusted Open Source License (CA-TOSL). CA lascia dunque intuire che saranno rilasciati ulteriori software con licenza Open Source, seguendo una strategia che dopo IBM, comincia a tentare anche Microsoft.

www.ca.com

CRYSTAL E OFFICE UN'ACCOPPIATA VANTAGGIOSA

All'ultimo TechEd americano, il 26 maggio scorso, Business Objects ha annunciato il rilascio di numerosi nuovi prodotti che si vanno ad integrare con l'offerta Microsoft per l'ufficio.
Di particolare rilievo, Crystal Enterprise Live Office: una soluzione che permette agli utenti di incorporare dati provenienti da Crystal Report all'interno di documenti Office.
I dati integrati in Word, Excel, Powerpoint o Outlook, saranno costantemente aggiornati, in modo da poter essere definiti "vivi": diventa così possibile garantire il continuo aggiornamento e la consistenza delle informazioni presenti nelle migliaia di documenti che circolano ogni giorno nelle aziende. Tutti i documenti correlati a dati prodotti attraverso Crystal Enterprise Live Office risulteranno allineati costantemente.
Nell'ambito delle integrazioni, non poteva mancare un prodotto che facesse esplicito riferimento a SharePoint Portal Server 2003: Crystal, nella versione 10, si inserisce ora perfettamente in SharePoint attraverso un kit di integrazione che consente l'accesso sicuro alle informazioni distribuite su diversi sistemi, ivi comprese applicazioni CRM (customer relationship management).

www.businessobjects.com

J2EE 1.5 OBIETTIVO SEMPLICITÀ

La facilità di sviluppo sarà al centro della versione 1.5 di Java 2 Enterprise Edition, successore dell'attuale J2EE 1.4.
Sul modello di J2SE 1.5, a breve disponibile in versione definitiva, J2EE 1.5 aiuterà i programmatori con il supporto dei metadati e dei generics.
I metadati consentiranno di migliorare la flessibilità delle applicazioni attraverso classi più facilmente integrabili in ambienti eterogenei.
Il supporto per XML ed UML diverrà parte integrante del linguaggio, andando a vantaggio proprio delle applicazioni enterprise cui J2EE è rivolto.
Anche l'introduzione dei generics migliorerà la flessibilità delle classi, consentendo di realizzare funzionalità che possano agire su molteplici tipi di dato.
Finora, J2EE 1.5 è stato spesso indicato come la versione Web Services oriented di J2EE, ma i miglioramenti apportati sottolineano quanto sia stata giudicata critica anche la facilità di sviluppo.
In questa direzione, Sun ha anche rilasciato Java Studio Creator; con questo ultimo rilascio lo scopo non dichiarato è di colmare il gap nei confronti della piattaforma .Net che, a dispetto di una base di installato molto più ridotta, ha nella semplicità il motivo di attrattiva più valido.

Webb.it 2004 Ecco com'è andata

Grande successo di pubblico, lo scorso Maggio per lo stand di ioProgrammo presente al Webb.it di Padova insieme al simpatico Sony Aibo.

di Alfredo Marroccelli

Lo stand di ioProgrammo è stata una delle mete preferite dalle scolaresche in uscita di istruzione!

Possiamo ben dire che la tre-giorni padovana di ioProgrammo al **Webb.it** sia stata una bella soddisfazione per chi scrive, una manifestazione di affetto e partecipazione da parte dei tantissimi lettori che hanno avuto piacere nel fermarsi nello stand della loro rivista di programmazione preferita. ioProgrammo è stata presente a Webb.it, la fiera-incontro dell'informatica svoltasi il 6, 7 e 8 Maggio.

Tra una evoluzione e l'altra dell'Aibo c'è chi ne approfitta per prendere una delle riviste omaggio.

L'occasione è stata la collaborazione con il progetto annuale sviluppato dallo staff di Webb.it, che quest'anno ha riguardato la programmazione dell'Aibo ERS-7, il simpatico robot-cagnolino di casa Sony, che in realtà è un vero e proprio concentrato di tecnologia, nonché l'oggetto del desiderio di ogni appassionato di robotica.
Il concorso promosso da ioProgrammo, proposto sul numero di Aprile, ha consentito a diversi lettori di provare il loro codice (l'Aibo è programmabile in C++, come forse già saprete dalle lezioni pubblicate in passato su questo argomento) e alle persone che popolavano lo stand di divertirsi guardando le evoluzioni del robottino.

I tre "addestratori" ufficiali dell'Aibo, nonché curatori dello stand, ne approfittano per una foto ricordo.

L'Aibo è in grado (nella versione "ludica" del suo software interno) di intrattenere chi lo osserva producendosi in danze ritmiche, accompagnate da una musichetta che egli stesso riproduce tramite uno speaker interno, giocare con il suo osso colorato, tirare calcetti alla sua pallina rosa e fare molte altre cose "intelligenti" tra cui, ad esempio, cercare autonomamente la basetta di carica quando le sue batterie sono agli sgoccioli.

Uno degli svariati tentativi di accensione del cluster di 4 XBox con Linux Debian, che faceva mostra di sé allo stand.

Prima di provare il codice sul robot, un lettore si fa immortalare con la sua rivista preferita.

È stata forse proprio la sua accattivante indole giocherellona a rendere una vera attrazione il nostro "Bubi", nome col quale è stato prontamente ribattezzato l'Aibo per potere facilmente rispondere ai comandi vocali che gli venivano impartiti (un vero tormentone il "Bubi, let's go dancing!" ripetuto con frequenza impressionante dai passanti che volevano vederlo ballare). Bubi è riuscito a strappare un sorriso anche ai più seriosi che si avvicinavano col sano scetticismo che contraddistingue chi del mondo dell'informatica ha fatto il suo mestiere. Desideriamo ringraziare di cuore

Alcuni lettori evidentemente avevano parecchi amici da convertire alla lettura di ioProgrammo, visto i pacchi da 20 riviste che portano!

tutte le persone che hanno contribuito alla buona riuscita di questa iniziativa e soprattutto i lettori, vecchi e nuovi, che sono venuti a trovarci. Alla prossima!

SOFTWARE SUL CD

Eclipse 3.0 M9

Lo stato dell'arte per lo sviluppo Java

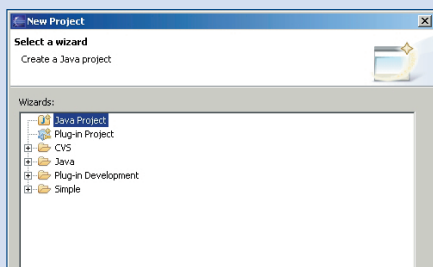
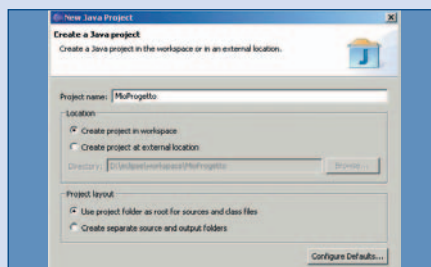
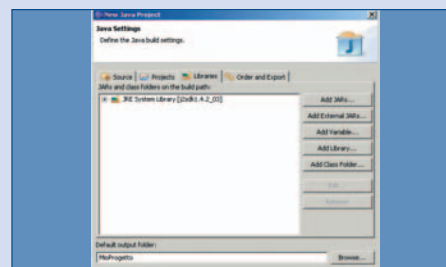
di Federico Mestrone

Sul CD allegato ad ioProgrammo con questo numero vi viene proposto un interessante tool di sviluppo Java dalla sorprendenti capacità. Si tratta del progetto Eclipse, inizialmente di proprietà IBM e da questa donato alla comunità open-source. Tale progetto consta in una raccolta di vari tipi di sottoprogetti che in realtà insieme formano, più che un software per lo sviluppo, un framework per la creazione di ambienti integrati di sviluppo per qualunque linguaggio e piattaforma. Ma la ragione per cui Eclipse sta avendo molto successo – che è anche la ragione per cui è a me particolarmente caro – sta appunto nell'IDE per la creazione di applicazioni Java, il quale, offerto di fatto come esempio di uso del framework che Eclipse mette a disposizione, è riuscito a imporsi come prodotto di alta qualità e di livello decisamente commerciale, pur essendo completamente gratuito. La nostra attenzione sarà quindi rivolta all'IDE Java di Eclipse, ed in particolare ci soffermeremo sulla versione 3 del prodotto: sul CD trovate la 3M9, ultima build disponibile al momento della pubblicazione della rivista. Si tratta di una versione stabile, perfettamente utilizzabile anche per ambienti di produzione, ma non ancora un rilascio definitivo di Eclipse 3. Presenta dei significativi miglioramenti rispetto all'attuale versione in rilascio ufficiale (2.1.3) in termini di semplicità d'uso, configurabilità e funzionalità offerte ed è per questo che ne parliamo in questa sede: tenete anche conto del fatto che non ci saranno differenze sostanziali rispetto al rilascio definitivo, se non bug-fix e ripulitura del codice.

Fig. 1: *La finestra di benvenuto di Eclipse 3.0 M9.*

INSTALLAZIONE E PRIMO UTILIZZO

Per cominciare, l'installazione del prodotto è veramente molto semplice. Dovete assicurarvi che sia installato sulla vostra macchina un JDK pari o superiore al 1.4.1, dopodichè potete scaricate il file .zip dal sito ufficiale *www.eclipse.org* (oppure evitate i tempi d'attesa e prendetelo dal CD!) ed estrarlo direttamente dove volete installare il prodotto finito! Tra i vari file estratti troverete l'eseguibile *eclipse.exe* da lanciare per fare partire l'IDE. All'avvio di Eclipse, dopo la richiesta della directory del workspace dove verranno tenuti i vostri progetti (lasciate pure il default), vi accoglierà la scherma-

CREARE E TESTARE UN'APPLICAZIONE JAVA CON ECLIPSE

1 Selezionate *File->New->Project* per creare un nuovo progetto. Dalla finestra *New Project* che vi si apre, scegliete poi *Java Project* e cliccate sul pulsante *Next* in basso sulla finestra per procedere al passo successivo.

2 Date un nome al vostro progetto. Questo sarà anche il nome della cartella dentro cui verrà tenuto il materiale che sviluppate. Lasciate le altre opzioni invariate. Premete il pulsante *Next.*

3 Qui potete specificare eventuali librerie aggiuntive per la compilazione del vostro progetto. Nel nostro caso, non c'è bisogno di indicare nulla: premete il pulsante *Finish*. Alla finestra di pop-up che chiede se passare alla prospettiva Java, dite di *sì.*

ta di benvenuto (la trovate in Fig. 1) con vari link ad elementi di aiuto e supporto per chi è nuovo all'ambiente. Chiudendo la finestra Welcome vi ritroverete invece nell'IDE vero e proprio. Eclipse tenta di agevolare il lavoro del programmatore organizzando i vari elementi offerti in prospettive *(perspective):* ogni prospettiva è un insieme di viste *(view),* cioè di finestre che offrono le più svariate funzionalità (explorer del repository di progetti, outline delle classi sviluppate, elenco dei package di un progetto, gerarchia delle classi, problemi di compilazione, etc), permettendo così al programmatore di lavorare secondo le sue esigenze: la prospettiva Java per scrivere il codice, quella Debug per fare il debug delle applicazioni, quella dedicata al team synchronising per lavorare con repository CVS, e via dicendo. Una piccola toolbar in alto a destra della finestra dell'IDE vi consente di aprire le varie prospettive e di passare tra una e l'altra.

UTILIZZARLO AL MEGLIO

Come buona parte degli ambienti di sviluppo disponibili, Eclipse propone un editor integrato con *syntax coloring,* completamente configurabile (menu *Window->Preferences,* poi *Java->Editor, pagina Syntax,* come in Fig. 2) ed una comodissima funzione di formattazione del codice, anch'essa completamente configurabile con un numero di opzioni impressionante (menu *Window->Preferences,* poi *Java->Code Style->Code Formatter,* create un profilo personalizzato e cliccate su *Edit,* come in Fig. 3) che prende tutto quello che avete scritto e lo sistema in base alle vostre scelte e alle convenzioni di stesura del codice previste da Java: basta premere *CTRL +SHIFT+F* durante l'editazione (mette in ordine indentazioni, spaziature, ritorni a capo, tutto insomma: è davvero ben fatto ed è una gran cosa per chi, come me, ci tiene ad avere il codice scritto in maniera pulita e leggibile).

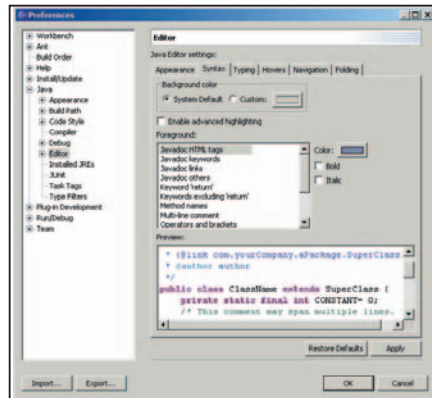



Fig. 2: **Le opzioni di syntax colouring offerte da Eclipse per l'editing Java.**

Un'altra comodissima ed innovativa caratteristica di Eclipse è quella di mostrare sempre tutti gli eventuali problemi di compilazione del vostro codice nella view dei problemi *(problems view),* grazie alla compilazione automatica delle classi che viene fatta ad ogni salvataggio del vostro lavoro (con *CTRL+S*). In questa maniera, ogni volta che salvate avete un immediato feedback sulla correttezza di quanto digitato: è quindi una buona idea abituarsi a salvare molto spesso, oltre che per evitare spiacevoli perdite qualora venisse a mancare la corrente, anche per verificare subito la reazione del compilatore a quello che abbiamo scritto.

UN VALIDO AIUTO

Per finire, non potendo in poco spazio elencare tutte le potenzialità di questo eccezionale prodotto, parliamo almeno di dove potete trovare aiuto: allo stesso livello degli ambienti di sviluppo che si pagano centinaia di dollari, Eclipse mette a disposizione una guida online completa e dettagliata delle sue caratteristiche.

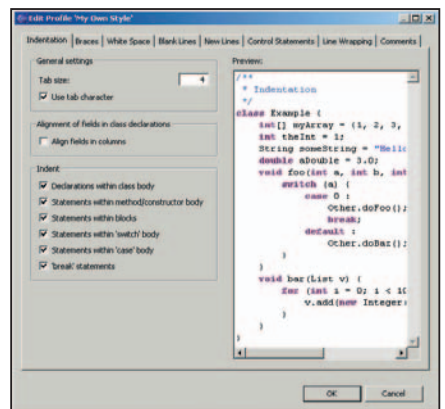
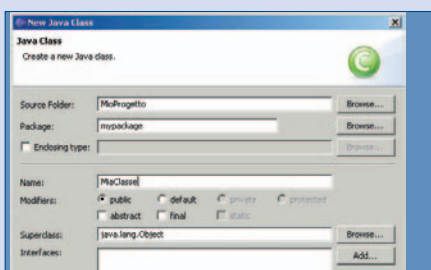
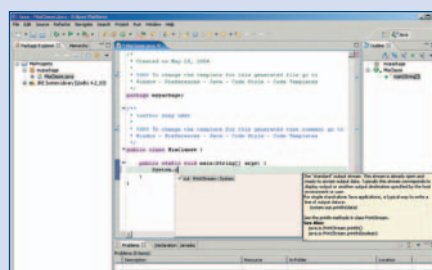
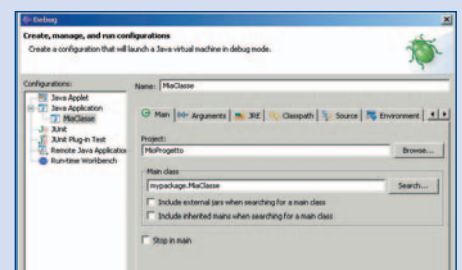



Fig. 3: **La finestra a più pagine per l'impostazione del code formatter.**

Se volete scoprire i trucchetti più interessanti per essere più produttivi, date un'occhiata ai *"tips and tricks"* (menu *Help-> Tips and Tricks* poi scegliete l'area d'interesse), mentre per il classico aiuto stile manuale, dal menu selezionate *Help-> Help Contents.*

☑ Eclipse 3.0M9
Produttore: consorzio eclipse.org
Sul web: www.eclipse.org
Prezzo: Gratuito
Nel CD: *Eclipse*

4 **Nella *view Navigator,* cliccate con il destro sul vostro nuovo progetto e selezionate il menu *New->Class.* Si apre così la finestra *New Java Class* dove potete definire una nuova classe Java per il vostro progetto. Selezionate la spunta per creare il metodo *main* e date un nome alla classe ed al suo package. Poi premete il pulsante *Finish.***

5 **A questo punto non resta che scrivere il codice della classe appena creata. Qui per esempio mettiamo un semplice *System.out.println* (Quello che voglio scrivere); all'interno del metodo main che è stato generato automaticamente, giusto per testare. A questo punto selezionate dalla barra del menu *Run->Debug* e vi si aprirà la finestra *Debug.***

6 **Selezionate *Java Application,* un clic su *New,* e verrà creata una nuova configurazione per l'esecuzione di applicazioni con metodo *main.* Selezionate il vostro progetto con *Browse* e lasciate che Eclipse trovi le classi con main, cliccando su *Search* (voi ne avete solo una!). Poi premete *Debug.* Il risultato dell'esecuzione della vostra classe nella view *Console.***

Eclipse ME

Il plug-in per Eclipse che consente di sviluppare in J2ME.

di Federico Mestrone

Tra il software a disposizione dei lettori questo bimestre, visto che è stato offerto Eclipse, recensito nelle pagine precedenti, abbiamo pensato di proporre anche un plug-in da installare sotto la sua piattaforma di sviluppo: EclipseME, un valido aiuto per chi si vuole cimentare nello sviluppo di applicazioni Java destinate ai dispostivi mobili, tipo i cellulari dotati di supporto per la piattaforma J2ME.

SVILUPPO MOBILE IN JAVA

Se vi ricordate (altrimenti potete riprendere l'articolo del numero 81 di ioProgrammo, a pagina 26, dedicato all'argomento), il J2ME è il framework di sviluppo proposto dalla Sun per lo sviluppo di MIDlet, termine con cui si definiscono le applicazioni che girano all'interno delle macchine virtuali Java predisposte per gli apparecchi wireless. Si tratta semplicemente di classi Java: potete immaginare questi MIDlet come se fossero degli applet ma, invece che essere eseguiti dentro un browser, essi hanno bisogno di un cellulare. I vari MIDlet che fanno parte di una stessa collezione applicativa sono raggruppati in un JAR associato ad un deployment descriptor (file di descrizione in formato XML) detto JAD, e nel complesso JAR e JAD sono chiamati MIDlet Suite. Da un altro lato, invece, dobbiamo parlare del concetto di plug-in di Eclipse. Il progetto Eclipse è un framework estensibile a mezzo di questi "plug-in", che possiamo definire come delle aggiunte al prodotto base che mettono a disposizione funzionalità e caratteristiche supplementari, per cui non sapremo mai dove finiscono le capacità di Eclipse, perché questo dipende dal numero e dalle funzionalità dei plug-in che vengono sviluppati per esso. Se volete farvi un'idea di cosa c'è a disposizione in giro per la rete, fate un salto all'indirizzo *www.eclipseplugincentral.com.*

UN PLUG-IN CHE SEMPLIFICA LE COSE

Quello che trovate sul CD allegato a questo numero, dunque, è un plug-in per Eclipse che permette a chi sviluppa di creare rapidamente lo scheletro di una applicazione J2ME. Un wizard specifico genera un progetto IDE che corrisponde ad una MIDlet Suite, con tanto di JAD editabile a mezzo di una finestra di proprietà, invece che direttamente dal codice XML, e un altro wizard genera invece la struttura basilare di una classe MIDlet, lasciando così a voi il solo compito di scrivere il codice per la funzionalità che intendete offrire. Alcuni menu contestuali che il plug-in aggiunge si occupano poi di generare quella combinazione JAR + JAD che può essere scaricata sul vostro cellulare, di eseguire i MIDlet sotto un emulatore di dispositivo mobile, ed altro ancora. Come ultima cosa, il plug-in si garantisce anche di fare le pre-verifica del codice Java: se ricordate dall'articolo sul J2ME menzionato più in alto, le macchine virtuali degli apparecchi wireless (le KVM) sono molto leggere e non si possono permettere di controllare l'integrità e validità del codice compilato, come fanno invece trasparentemente le JVM standard: è necessario perciò che il codice venga preverificato prima dell'installazione ed esecuzione, aggiungendo carico al lavoro del programmatore. Ma come già detto, EclipseME fa tutto questo per voi.

LA PRIMA APPLICAZIONE

Vediamo adesso, a semplici passi, cosa fare per renderci operativi come sviluppatori J2ME. Partirò dal presupposto che Java ed Eclipse siano già installati. Dobbiamo allora installare il JDK per J2ME, che ci metterà a disposizione le interfacce per lo sviluppo ed un emulatore di cellulare di test: scaricate il prodotto dalla Sun all'indirizzo *http://java.sun.com/products/j2mewtoolkit/download-2_1.html,* ed eseguite il file ricevuto. La procedura di installazione dovrebbe rilevare automaticamente il JDK che avete e vi permetterà di scegliere la directory dove installare il Wireless *Development Toolkit:* lasciate quella di default, C:\WTK21. Ad installazione avvenuta, possiamo passare ad EclipseME.

1) Scaricate il plug-in dal sito *http://eclipseme.sourceforge.net,* oppure utilizzate la versione che trovate sul CD allegato ad ioProgrammo. Per installare il prodotto, dovete soltanto estrarre il file compresso scaricato dentro la cartella plugins che trovate sotto la directory di installazione di Eclipse. Alla fine dell'estrazione, dentro plugins dovrete avere una cartella *eclipseme_0.4.0.* Se la versione che avete voi fosse una successiva alla 0.4, ovviamente il nome della directory sarebbe differente!

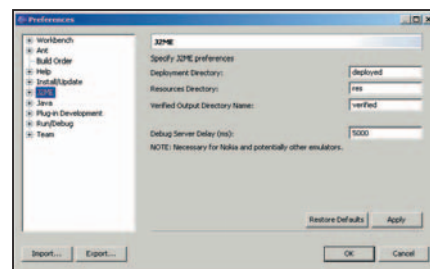



Fig. 1: Il plug-in è installato correttamente!

2) Una volta (ri)avviato Eclipse dopo l'aggiunta del plug-in, controllate se l'installazione è stata portata a termine correttamente: aprite la pagina delle preferenze *(Window->Preferences…)* e verificate di avere una nuova voce J2ME nella lista delle opzioni, come vedete in Fig. 1. Questa voce, aggiunta da EclipseME, è segno che adesso il plug-in è installato ed attivo. Restate in questa finestra.

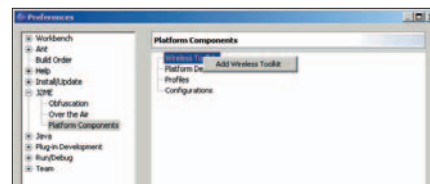



Fig. 2: Indichiamo la posizione del toolkit.

3) Dobbiamo adesso indicare ad EclipseME dove si trova il nostro Wireless Toolkit. Per fare questo, passate alla pagina dei toolkit cliccando sulla voce *Platform Components,* cliccate poi col destro su *Wireless*

Toolkits per mostrare il menu contestuale *Add Wireless Toolkit,* come vedete in Fig. 2. Selezionando la voce di tale menu contestuale, vi verrà richiesta la directory dove avete installato il prodotto della Sun, che per default è *C:\WTK21,* ed EclipseME identificherà così il toolkit per lo sviluppo J2ME (come in Fig. 3). Premete il pulsante *Finish* sulla finestra *Add Wireless Toolkit,* e poi il pulsante *Ok* sulla finestra *Preferences.* Ora siamo pronti per sviluppare il nostro primo MIDlet.

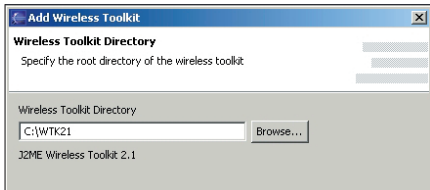



Fig. 3: ***Il toolkit standard di Sun.***

4) Dovreste essere adesso di nuovo nel *workbench* (la finestra principale). Dalla barra del menu in alto, aprite il menu *File->New->Project…* e dalla finestra *New Project* selezionate la voce *J2ME MIDlet* Suite (la vedete in Fig. 4, è stata aggiunta alla lista grazie al plug-in appena installato). Cliccate su *Next* e date un nome al vostro progetto, per esempio *"MiaMidletSuite".* Ora cliccate su *Next* e scegliete una piattaforma MIDP per lo sviluppo (vedi Fig. 5, dove viene scelto *MIDP 2.0*). Nella pagina che segue (Fig. 6), potete aggiungere eventuali librerie per la vostra applicazione J2ME. Notate che è stato impostato un filtro che non mostra il materiale di lavoro del plugin (directory *verified* e *deployed*), rendendo così più pulita la lista di file con cui il programmatore avrà a che fare nelle varie view.

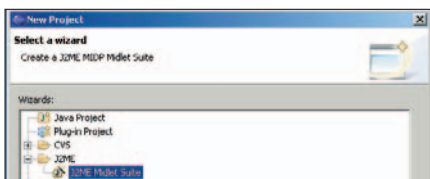



Fig. 4: ***Il primo passo per costruire un'applicazione.***

5) Adesso, dalla view *Package Explorer* (se non la vedete, aprite la *prospettiva Java*), cliccate col destro sul nuovo progetto J2ME che avete creato e selezionate *New->Other…* per far comparire la finestra *New* della Fig. 7. Da questa finestra scegliete la voce *J2ME MIDlet,* anch'essa parte del plugin EclipseME. Cliccando *Next* vi si aprirà la finestra di creazione di un nuovo MIDlet, che trovate in Fig. 8. Qui dovete solo dare un nome al package e alla classe che volete creare ed indicare che la vostra classe implementerà l'interfaccia *CommandListener* (cliccate sul pulsante Add, nella finestra che compare scrivete *"CommandListener"* e cliccate *Add* e *Close*): le altre opzioni possono essere lasciate al loro default. Cliccando su *Finish* verrà generata una nuova classe per il vostro MIDlet e la stessa sarà aperta nell'editor per il vostro sviluppo.

6) Alla classe appena creata aggiungete i

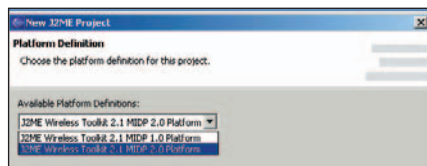



Fig. 5: ***La scelta della piattaforma.***

seguenti statement di import:

```
import javax.microedition.lcdui.Screen;
import javax.microedition.lcdui.TextBox;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.TextField;
```

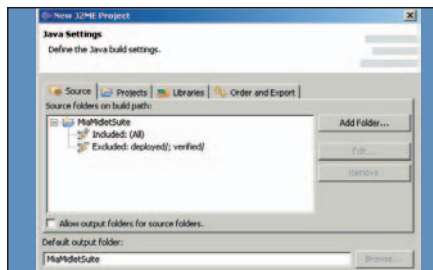



Fig. 6: ***L'ossatura della Midlet Suite è pronta.***

7) Aggiungete poi la seguente variabile di istanza

```
private Command cmdExit = new
        Command("Esci",Command.EXIT,0);
```

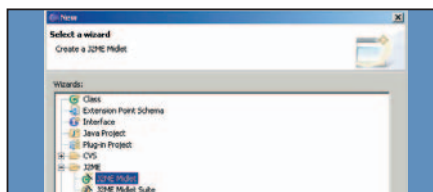



Fig. 7: ***E' il momento del Midlet.***

8) Implementate il metodo startApp con questo codice

```
Screen myScreen = null;
Display d = Display.getDisplay(this);
if (d.getCurrent() == null) {
    myScreen = new TextBox("Welcome",
        "Questo è il mio primo MIDlet!", 100,
        TextField.ANY);
    myScreen.addCommand(cmdExit);
    myScreen.setCommandListener(this);
    d.setCurrent(myScreen);}
```

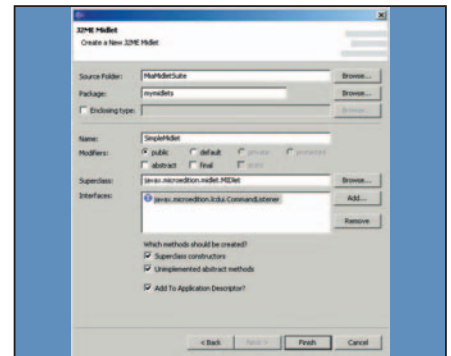



Fig. 8: ***L'ultimo passo nel Wizard.***

9) Infine inserite il codice seguente come implementazione del metodo *commandAction*

```
if ( arg0 == cmdExit ) {
  try {
      destroyApp(false);
      notifyDestroyed();
  } catch (MIDletStateChangeException msce)
{}
}
```

Fig. 9: ***Il nostro Midlet in azione.***

Ora non resta che eseguire il nostro codice in un emulatore di cellulare messo a disposizione dal Wireless Toolkit ed utilizzabile tramite EclipseME. Dopo aver selezionato la classe MIDlet che abbiamo terminato di scrivere, cliccate sul menu *Run->Debug as->Emulated J2ME Midlet* e dopo qualche secondo si aprirà un cellulare virtuale come quello della Fig. 9, con il vostro MIDlet in esecuzione.

Ace OS 2.0 Beta

Il sistema operativo Open Source tutto da e studiare.

Ace OS è un nuovo sistema operativo libero distribuito con licenza GNU GPL. Lo scopo degli sviluppatori è principalmente quello di creare un sistema operativo per fini didattici e di ricerca, per far sì che si possano “studiare” la struttura ed i meccanismi che agiscono all’interno di un sistema operativo e del suo componente principale, il kernel. Il progetto, pur essendo stato avviato da qualche anno, è ancora nella fase iniziale di sviluppo, non può essere certamente considerato un sistema operativo completo nel senso stretto del termine, ne tanto meno può essere paragonato a sistemi operativi come MS Windows e GNU/Linux, sia per quanto riguarda la complessità strutturale che le funzionalità disponibili. Proprio grazie alla sua compattezza, semplicità e disponibilità del codice (grazie alla licenza Open Source) è possibile analizzare e modificare i sorgenti (Assembly, C e C++) per sperimentarne il funzionamento e, magari, può essere utile come punto di partenza per la realizzazione di un nuovo sistema operativo per macchine con architettura x86 (*http://aceos.netfirms.com*).

REQUISITI HARDWARE E SOFTWARE

Il sistema operativo completo, in formato zip, occupa circa 230 KB di spazio e certamente per essere eseguito non necessita di configurazioni hardware esasperate, comunque il sistema minimo deve essere composto da una CPU 486, 4 MB di RAM e supporto PCI (*http:// aceos.netfirms.com/SystemRequirement.html*). Questo è quanto consigliato dall’autore, ma probabilmente pochi saranno interessati ad installare realmente il sistema; per questo, sia sul sito del progetto che nell’esempio seguente, eseguiremo Ace OS in emulazione mediante il software Bochs. Quest’ultimo sarà indispensabile, successivamente, perché consentirà di effettuare il debug del codice dopo aver apportato qualche modifica. Per quanto riguarda i requisiti software, sono necessari alcuni strumenti di sviluppo per compilare il sistema operativo, i cui sorgenti sono disponibili in formato Assembly, C e C++ (*http://aceos.netfirms.com/DevelopmentKits.html*). All’indirizzo precedente è presente la lista completa dei programmi, per “fortuna” non tutti necessari e comunque disponibili insieme al sistema operativo sul CD allegato alla rivista.

ISTRUZIONI PER COMPILARE ACE OS

Prima di procedere, bisogna installare il compilatore MingGW necessario per compilare i sorgenti C/C++, il compilatore NASM per i sorgenti in Assembly, un buon editor per programmatori (SciTe) e l’emulatore Bochs.
Per chi lo desidera, è possibile installare alcuni IDE *(Integrated Development Environment)* o ambienti di sviluppo integrati che semplificano la gestione e la compilazione del codice: DevC++ è un ambiente di programmazione avanzato per codice C/C++ dotato di interfaccia grafica e di tutte le funzionalità comuni agli strumenti di sviluppo più moderni; *NasmIDE* è un IDE a riga di comando per il compilatore NASM. Dopo aver estratto il contenuto del file *AceOS_S2.0B.zip* e installato gli strumenti contenuti nel file tools.zip, copiamo la directory *AceOS_S2.0B* in un posizione qualsiasi e rinominiamola con il nome *aceos*.
Eseguiamo una shell DOS *(Start/Esegui/Command.com)* e spostiamoci nella directory *aceos*. Da qui digitiamo il comando *mingw32-make* e premiamo *Invio*; questo compilerà tutti i sorgenti creando numerosi file oggetto linkandoli tutti nel file *Kernel.bin*. Se la compilazione avviene con successo sulla console DOS dovrebbero apparire i messaggi seguenti:

ld -Ttext 0x19000 -Tdata 0x258c0 --image 0x19000 --shared -entry KernelEntry -nostartupfiles -o Kernel.exe

Kernel.o Processor.o Timer.o CTimer.o CKernel.o MemMan.o Heap.o DeviceDriver.o IO.o TMan.o TaskMan.o MemMove.o

VGAText.o VGACursor.o Keyboard.o CKeyboard.o PIC.o PCI.o RTC.o GSH.o Harddisk.o HMess.o Printf.o Partition.o

FAT.o String.o Beep.o -LC:\MinGW\lib\gcc-lib\mingw32\3.2 -LC:\MinGW\lib -lmingwex –lgcc

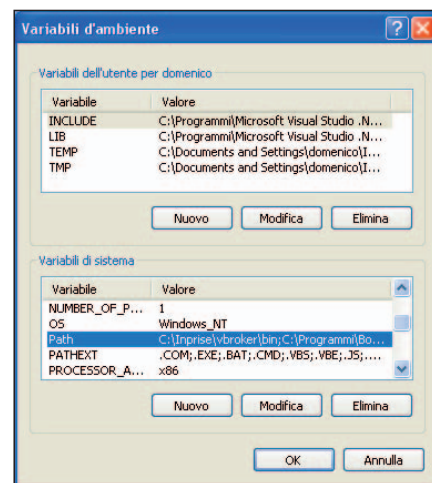



Fig. 1: Finestra di dialogo per la gestione delle variabili d’ambiente.

Prima di eseguire il comando *mingw-32-make*, verificate che la variabile di ambiente *PATH* contenga il percorso alle directory degli strumenti di sviluppo, altrimenti bisogna aggiungerli. Per fare ciò in Windows XP basta selezionare dal Menu di avvio *Start/Pannello di controllo/Sistema*, da qui basta selezionare la scheda *Avanzate* e successivamente il pulsante *Variabili d’ambiente* (Fig. 1).

CREAZIONE DEL FLOPPY DI BOOT

Per creare il dischetto di avvio, spostiamoci nella directory *Boot* ed eseguiamo il comando *make*, dopo di che inseriamo un floppy nel drive A ed eseguiamo il comando *mkboot.bat*. Questo servirà a creare il disco di boot per avviare il sistema operativo compilato in precedenza.

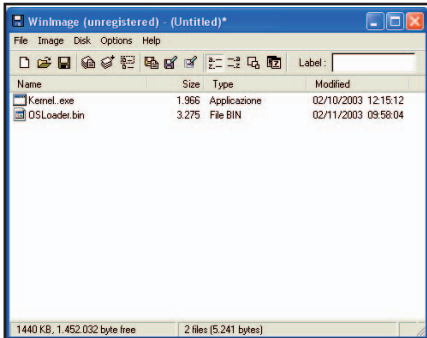



Fig. 2: ***Creazione del file immagine con WinImage.***

ACE OS IN EMULAZIONE CON BOCHS

Per completare i passaggi successivi è necessario disporre del software di emulazione Bochs (fornito nel CD) e del programma WinImage. Di quest'ultimo è possibile ottenerne un copia in prova (shareware) per 30 giorni scaricandola dall'indirizzo *www.winimage. com.* Avviamo *WinImage* e creiamo una nuova immagine selezionando *File/New* dal menù principale. Successivamente, premendo il tasto *INS* (oppure selezionando *Image /Inject),* inseriamo i file *Kernel.exe* e OSLoader.bin (Fig. 2).

```
Bochs for Windows    [F12 enables mouse]
Ace OS V2.0.0
Initializing Timer :: Sucess
GenuineIntel CPU Family 5 Model 1 Steping 3.
Virtual Page Size = 4096. Physical Memory 8 MB RAM installed.
Real Time Clock Settings 11-11-3
Initializing Keyboard :: Sucess
Detecting PCI Devices :: 0 device(s) found
Detecting ATA Devices ::  2 device(s) found
 Master Device ( H0 ) :: Generic 1234          - N/A
 Cylinders 40 Heads 16 Sectors 63. LBA Sectors 40320
 Slave  Device ( H1 ) :: Generic 1234          - N/A
 Cylinders 20 Heads 16 Sectors 63. LBA Sectors 20160
Initializing GFS :: Sucess
Initializing IFS :: Sucess
 FAT12 is found on H\0\0. Assigning drive letter C: ... done
 FAT12 is found on H\1\0. Assigning drive letter D: ... done
Initializing FAT File System :: Sucess
File System(s) Registered ::
 FS Type [I] - [Installable FS] Total Volumes 1
 FS Type [F] - [FAT File System] Total Volumes 2
 success
 Loading CShell.exe... :
 done
 CShell Version 0.1
```

Fig. 4: ***Ace OS in emulazione con Bochs.***

Selezioniamo *Image/Boot Sector Properties* e, dalla finestra di dialogo che appare, premiamo il pulsante *Open,* carichiamo il file *Boot.bin* dalla directory *aceos/Boot* e infine click sul tasto *Save.*
A questo punto non ci resta che salvare l'immagine appena creata e avviare Bochs. Dalla schermata principale selezionare *(2) Read options from...* e indicare il file di configurazione: è possibile inserire il nome del file *bochsrc.txt* oppure premere semplicemente *Invio.*
Alla schermata successiva basta selezionare *(3) Edit options,* in seguito *(8) Disk options,* scegliere *(1) Floppy Disk 0* e indicare il nome e il percorso completo del file immagine creato in precedenza. I passaggi successivi servono per configurare il BIOS da utilizzare per il sistema e la scheda video, rispettivamente *BIOS-BOCHS-LATEST* e *VGA-BIOS-elpin-2.40 BIOS-BOCHS-LATEST.* Per selezionarli bisogna tornare al passaggio precedente scegliendo l'opzione *0 (zero)* e in questa schermata basta scegliere *(6) Memory options* e successivamente indicare il percorso corretto dei due file posizionati nella directory di installazione di Bochs.
A questo punto bisogna tornare indietro, fino alla schermata iniziale, utilizzando l'opzione *0 (zero),* salvare la configurazione con *(4) Save options to…* e successivamente avviare l'emulazione selezionando l'opzione *(5) Begin Simulation.*
L'aspetto più interessante di Ace OS è la possibilità di studiare e modificare il codice sorgente e, grazie all'emulazione con Bochs, verificarne immediatamente i cambiamenti. Infatti, è possibile effettuate il debug del codice, avviando Bochs in modalità debug mediante la shell DOS con il comando *bochsdbg.* Prima di eseguire il comando è necessario modificare la variabile d'ambiente *PATH* affinché contenga il percorso completo alla directory di installazione di Bochs.

```
Bochs 2.1.1
========================================================================
                       Bochs x86 Emulator 2.1.1
                          February 08, 2004
========================================================================
------------------------------
Bochs Configuration: Main Menu
------------------------------

This is the Bochs Configuration Interface, where you can describe the
machine that you want to simulate.  Bochs has already searched for a
configuration file (typically called bochsrc.txt) and loaded it if it
could be found.  When you are satisfied with the configuration, go
ahead and start the simulation.

You can also start bochs with the -q option to skip these menus.

1. Restore factory default configuration
2. Read options from...
3. Edit options
4. Save options to...
5. Begin simulation
6. Quit now

Please choose one: [2] _
```

Fig. 3: ***Shermata iniziale di Bochs.***

☑ Ace OS 2.0 Beta

Autore: samuelhard@yahoo.com
Sul Web: http://aceos.netfirms.com/
Licenza: GNU GPL
Nel CD: ***AceOS_S2.0B.zip***

SOURCE FORGE

Bochs-2.1.1

Emulatore di sistemi con architettura x86

Bochs è un emulatore Open Source altamente portabile, per PC con architettura IA-32 (x86) scritto in C++, in grado di funzionare sulla maggior parte delle piattaforme. Include l'emulazione delle CPU x86 (Intel, AMD, Ciryx, etc), dei dispositivi di I/O più comuni, e dispone di un proprio BIOS. Attualmente, Bochs può essere compilato per emulare CPU 386, 486, Pentium, Pentium II e Pro, comprese le istruzioni MMX , SSE, SSE2 e 3DNow (AMD). Bochs è capace di eseguire la maggior parte dei sistemi operativi all'interno dell'emulatore: Linux, Windows 95, DOS e Windows NT 4. Bochs è stato scritto da Kevin Lawton ed attualmente è sviluppato dal team Bochs project ed è disponibile sul sito Internet "*http://bochs.sourceforge.net*".

bochs-2.1.1.zip

EasyPHP 1.7

Applicazioni web con PHP e MySQL

EasyPHP installa e configura automaticamente un ambiente di sviluppo completo per la creazione di applicazioni web server side che interagiscono con i database, utilizzando il linguaggio di scripting PHP, il server web Apache e il database server MySQL. Inoltre, include il programma *phpMyAdmin*, un'applicazione web based che semplifica notevolmente la gestione di database MySQL grazie ad un'interfaccia semplice ed intuitiva. La nuova versione 1.7 include i seguenti software: Apache 1.3.27, PHP 4.3.3, MySQL 4.0.15 e phpMyAdmin 2.5.3.

easyphp1-7.zip

Firebird 1.5

Il database relazionale del nuovo millennio

Il motore del database Firebird è stato creato da un team indipendente di sviluppatori volontari partendo dal codice sorgente del database InterBase prodotto da Borland (allora Inprise) e distribuito con licenza IBL (InterBase Public License) il 25 Luglio 2000. Firebird è un database relazionale (RDBMS) che supporta in modo quasi completo lo standard ANSI SQL-92, ed è disponibile per sistemi GNU/Linux, Windows e può essere eseguito su una varietà di piattaforme UNIX. Inoltre, Firebird offre prestazioni elevate e pieno supporto per stored procedures e triggers.

firebird-1.5.0.zip

jEdit 4.1

Editor Java multipiattaforma

Realizzato completamente in Java, jEdit è un completo editor di testi per programmatori, disponibile per numerose piattaforme tra cui MacOS X, OS/2, GNU/Linux (Unix) e Windows. Il programma, per l'immediatezza e la semplicità di utilizzo, è molto usato per scopi didattici e da programmatori non professionisti. jEdit integra un completo linguaggio di macro e dispone di un'architettura facilmente estensibile mediante l'utilizzo di numerosi plugin facilmente gestibili tramite il "plugin manager".

jedit41.zip

QCad 2.0.3.1

Il disegno tecnico 2D diventa un gioco

QCad è una potente applicazione CAD (Computer Aided Design) per la creazione di disegni tecnici 2D per costruzioni, interni e addirittura parti meccaniche. Inoltre, riconosce i formati CAD più comuni come DFX (importazione ed esportazione), HPGL, DGN e EPS (solo importazione).

qcad_2_0_3_1.zipa

TightVNC 1.2.9

Controllo remoto mediante VNC

VNC (abbreviazione di Virtual Network Computing) è un sistema client/server che permette l'accesso remoto a desktop grafici. Con VNC, è possibile accedere ad un sistema da qualunque altra macchina connessa ad Internet o in una rete locale . VNC è software libero (rilasciato sotto la GNU General Public License) ed è disponibile per numerose piattaforme tra cui Windows e GNU/Linux. TightVNC è una versione evoluta del VNC originale che include molte nuove caratteristiche, migliorie, ottimizzazioni e correzioni di bug. Inoltre, può essere usato per l'amministrazione remota di Windows, Unix o ambienti di rete misti.

tightvnc-1.2.9.zip

Relo 0.9.9

IDE per lo sviluppo di applicazioni C/C++

Ambiente di sviluppo integrato (IDE) per creare applicazioni Windows in linguaggio C/C++. Il software distribuito con licenza Open Source (GNU GPL) è disponibile sottoforma di sorgenti o eseguibile per piattaforma Windows. Relo utilizza i compilatori MingW32 (la versione per Windows del celebre compilatore GCC - GNU Compiler Collection) e Borland C++. L'ambiente di sviluppo è intuitivo, leggero e altamente personalizzabile. Inoltre, consente di creare codice per oltre 19 linguaggi di programmazione con funzioni di syntax highlighting, e dispone di strumenti per importare progetti da altri ambienti di sviluppo come Visual C++, Borland C++ Builder, Dev C++ e Delphi.

relosetup099.zip

Zope 2.7

Application server professionale

Zope è una piattaforma che permette a sviluppatori con differenti livelli di competenza di costruire applicazioni web ad alto contenuto: ogni collaboratore ha a disposizione gli strumenti per creare e gestire personalmente il sito, scrivendo articoli, proponendo news, documentazione, etc. In termini tecnici Zope è un framework, costituito da moduli che interagiscono tra di loro per fornire numerosi servizi: server web, database ad oggetti, una completa interfaccia di amministrazione, interazione con database relazionali e supporto per numerosi linguaggi di scripting. Il linguaggio nativo di Zope è Python, ma è possibile utilizzare anche Perl. Zope è distribuito con licenza Open Source ZPL (Zope Public License). I termini della licenza ZPL consentono di ottenere e modificare il codice sorgente.

zope-2.7.0.zip

AppForge Crossfire

Sviluppo Mobile con VB 6 e VB.NET

Con Crossfire, puoi riversare tutta l'esperienza maturata lavorando con VB 6 o con VB.NET nello sviluppo di applicazioni per dispositivi mobili. Crossfire si integra perfettamente sia nell'IDE di Visual Studio .NET che in quello di Visual Basic 6.0, consentendo di sviluppare in pochissimo tempo applicazioni che girano sulla maggioranza dei dispositivi mobili esistenti: alm OS, Pocket PC 2000/2002, Windows Mobile 2003, Nokia Series 60, e Symbian UIQ. Sviluppando applicazioni con Crossfire potremo approfittare di numerose agevolazioni, a partire dalla dimensione delle form già settata per lo specifico dispositivo su cui vogliamo distribuire l'applicazione. Nella toolbar avremo a disposizione dei nuovi controlli appositi per applicazioni Mobile ed un nuovo menu con decine di nuove funzionalità. Per registrarsi occorre seguire il link: *http://scripts .appforge .com /evals /afevals.aspx?p=AFCF* e ciccare sul tasto *"Request evaluation key only".*

Crossfire_5.0.1.exe

FLASH MX 2004

In italiano, la versione completa del più celebre applicativo per la creazione di progetti multimediali

Due versioni, una dedicata ai designer (Flash MX 2004) ed una pensata per i developer (Flash MX 2004 Professional). Questa scelta di realizzare due pacchetti distinti testimonia la divisione di ruoli che la Macromedia ha individuato nell'utilizzo della tecnologia alla base di Flash, e che possiamo definire l'evoluzione naturale di una strategia di cui si potevano cogliere i primi segnali a partire dalla versione MX. Forte integrazione con XML, utile sia nella manipolazione dei dati che nella gestione dei componenti. La versione *Professional* possiede tutte le caratteristiche della versione normale con in più dei componenti aggiuntivi creati per agevolare lo scambio di dati e numerose opzioni, molte delle quali pensate per dialogare con programmi di terze parti.

FlashMX2004-it.zip

SourceCode to Flowchart 2.43

Analizza il tuo codice sorgente attraverso flowchart

Avete mai provato a sviluppare un'applicazione dopo un mese (o un anno) di pausa? Se la risposta è affermativa, avrete probabilmente provato una forma di disorientamento simile a quella che si prova mettendo mano al codice di un'applicazione sviluppata da altri. In entrambi i casi, ci sarà utilissima l'utility che presentiamo: a partire dal codice, SourceCode to Flowchart genera un diagramma di flusso che consente di capire esattamente il percorso seguito all'interno dell'applicazione. Buona l'interfaccia, mentre risulta eccellente l'ampiezza dei linguaggi supportati: C, C++, VC++(Visual C++ .NET), VB(Visual Basic), VBA, Qbasic (quickbasic), VBScript(VBS), ASP, Visual C#, Visual Basic .NET,Visual J# .NET, VC++.NET, ASP.NET, Java, JSP, JavaScript, Delphi(Object Pascal), PowerBuilder, PHP, Visual FoxPro e Perl. Versione di valutazione: risulta limitato il livello di analisi consentito.

s2f243.zip

XMLSpy Home Edition 2004r4

Scegli lo sviluppo XML che fa per te

Questo mese, ioProgrammo vi offre due versione di XMLSpy: la Home la Enterprise. La Home edition è gratuita e si presenta come un tool entry level per lo sviluppo di documenti XML. Ideale per hobbisti e studenti, rappresenta uno dei punti di riferimento per gli sviluppatori "casalinghi". XMLSPY Enterprise Edition è tra i migliori tool disponibili: agevola la costruzione di applicazioni XML-based, siti Web, Web Services e tutto quanto ruota attorno all'XML, la stella incontrastata dell'attuale panorama della programmazione. Alla fine dell'installazione è possibile scaricare automaticamente dei alcuni tool aggiuntivi. Al primo avvio ci viene richiesto il nostro nome e un indirizzo mail cui verrà spedita in pochi, istanti, la chiave di attivazione necessaria ad attivare il software per trenta giorni.

XMLSPYHomeComplete2004.exe
XMLSPYEntComplete2004.exe

Mapforce Enterprise Edition 2004r4

Potente tool per l'integrazione di dati

Una volta "disegnato" lo schema di trasformazione che vogliamo realizzarem Mapforce si fa carico di generare il codice (Java, C++, C# o XSLT) necessario a completare l'opera. Le trasformazioni supportate sono XML-to-XML e Database-to-XML: risparmio di tempo e garanzia dei risultati sono i principali vantaggi di una soluzione di questo tipo. Attraverso Mapforce 2004 è possibile caricare direttamente le tabelle del database e gli schemi XML, per poi disegnare visualmente la mappa che trasforma le tabelle in un qualsiasi modello di dati espresso nell'XML Schema. Fatto questo, Mapforce 2004 genera automaticamente il codice applicativo necessario ad effettuare la conversione dal database indicato verso l'XML. Il codice così ottenuto è completamente customizzabile e supporta tutti i più diffusi database: SQL Server, Oracle9i e qualsiasi database per cui sia disponibile un driver ODBC. Versione di valutazione valida trenta giorni.

MapForceEntComplete2004.exe

Java 2 SDK, Standard Edition 1.4.2

Tutto quello che serve per realizzare applicazioni Java

L'ambiente di sviluppo Sun che negli ultimi anni si è imposto come la prima scelta per i programmatori che lavorano in ambito multipiattaforma. In questa versione, nuove funzionalità e migliori prestazioni arricchiscono l'ambiente. Rich client application e Web Services sono i campi in cui sono più evidenti i miglioramenti. Tra i miglioramenti che più faranno gola agli sviluppatori ci sono sicuramente quelli inerenti Swing. Davvero ghiotti i due nuovi look&feel: GTK+ e, finalmente, Windows XP. Anche le applicazioni Java possono così integrarsi pienamente nell'ambiente visuale del più recente Windows. Anche GTK+ risulta molto interessante: attraverso un semplice resource file, è possibile settare i parametri fondamentali del look&feel. Sempre in ambito Swing, è da notare la pesante cura dimagrante cui è stata sottoposta JFileChooser, che risulta essere molto più veloce della precedente versione, in alcuni casi anche 300 volte più veloce!

j2sdk-1_4_2_04-windows-i586-p.exe

Installer2Go 4.0.3

Un sistema facile e gratuito per realizzare pacchetti di installazione

Un tool per la creazione di file autoinstallanti che non impegna lo sviluppatore nel dover imparare l'ennesimo linguaggio di scripting: con una interfaccia che punta tutto sul frag&drop, realizzare pacchetti di installazione diventa un vero gioco da

ragazzi. Benché gratuito, Installer2Go va incontro a tutte linee guida per la certificazione WindowsXP. Versione dimostrativa: alla fine di ogni istallazione si è rimandati ad una pagina pubblicitaria che fa riferimento al produttore di Installer2Go.

i2g.exe

Astrum InstallWizard 2.02.7

Crea il tuo setup

Astrum InstallWizard è un versatile software per la creazione di pacchetti di installazione. Potente e ben ideato, consente di arrivare al pacchetto di installazione completo attraverso una semplice e chiara procedura guidata. L'ottimo help ed i numerosi tutorial aiutano anche i meno esperti a realizzare in pochi istanti pacchetti di livello professionale. Non rinuncia alla completezza comprendendo il supporto per file JPEG ed MP3. E' possibile utilizzare variabili utente, dividere I file di installazione su più dischi e interagire in vario modo con il registro di windows. Semplice ed altamente personalizzabile. Da rimarcare la presenza di un apposito Wizard per la creazione di update. Versione dimostrativa, inibita la possibilità di creare distribuzioni.

aiw.exe

XStudio - Standard Edition 2.5

Semplifica la scrittura di codice XSL

Un grande ausilio alla produzione di documenti XSL: attraverso l'ampio uso del drag& drop, XStudio mette anche i meno esperti nelle condizioni di poter generare trasformazioni da XML a HTML e da XML a XML. L'interfaccia visuale non richiede alcuna conoscenza di XSL. Versione di valutazione valida trenta giorni.

XStudio.zip

SheerPower 4GL 3.5

Crea applicazioni per Windows in un batter d'occhio!

Il linguaggio adottato da SheerPower 4GL è particolarmente semplice da utilizzare e mutua dai linguaggi di scripting la loro estrema immediatezza. Per quanto leggero, l'IDE si presenta completo e disponibile allo sviluppo di applicazioni di qualsiasi dimensione. Oltre ad un motore di database integrato, SheerPower offre anche l'indispensabile supporto per ODBC. Anche le funzioni di accesso ad Internet risultano particolarmente curate. Gratuito.

sp_install.exe

Code Co-op 4.1

Funziona in modalità distribuita senza la necessità di utilizzare un server!

Code Co-op 4.0 è un programma di controllo versioni sorgenti per applicazioni realizzate in C++, Java, HTML… Può funzionare in modalità distribuita senza la necessità di utilizzare un server. Supporta non solo la collaborazione tramite Lan, ma anche via e-mail. In tal modo, è possibile collaborare ad un progetto pur non risiedendo nella stessa area geografica e, soprattutto, senza la necessità di installare un server! Grazie all'uso di appropriati messaggi di posta elettronica, è possibile ottenere la sincronizzazione dei sorgenti persino tra postazioni che non sono costantemente connesse in rete. Si integra con strumenti di sviluppo quali Microsoft Visual Studio .NET, Borland Delphi, C++ Builder ed altre applicazioni che utilizzano le API SCC della Microsoft. Versione di valutazione valida trentuno giorni. Per scegliere di utilizzare la versione trial, basta cliccare su "Keep evaluating" e verrà visualizzata la finestra principale.

co-op.exe

ShareGuard Copy Protection 2.1

Aumenta la protezione del tuo software

Un tool per proteggere le versioni shareware delle applicazioni che distribuiamo, rendendo inutili le copie non licenziate. Potremo realizzare e distribuire versioni shareware delle nostre applicazioni, decidendo la durata del periodo di prova dopo il quale l'applicazione diventerà inservibile. Funziona su piattaforma Windows e con qualsiasi ambiente di programmazione (Visual Basic, MS Access, C++, Delphi, VB.NET, C#). Versione dimostrativa (Ovviamente, visto il genere!) valida trenta giorni e limitata a cinquanta utilizzi.

ZSSHGD.zip

CodeLogic 1.5

Non solo UML

CodeLogic fornisce una dettagliata analisi di qualsiasi progetto Java, generando diagrammi UML (di classe e sequence) relativi alle classi e ai metodi selezionati. Uno degli aspetti più interessanti è la possibilità di visualizzare il codice attraverso una particolare interfaccia grafica che, a fianco di ogni riga, riporta un simbolo che ne semplifica la comprensione. Questa visualizzazione si può considerare come diretta conseguenza dei digrammi sequence e agevola notevolmente la lettura del codice. Impagabile nel caso in cui vi troviate a studiare codice prodotto da altri. Versione di valutazione valida quindici giorni e limitata ad un utilizzo continuo di massimo dieci minuti.

codelogic_setupWin.exe

Serial Basic 2.2

Tutta la libertà di programmare la porta seriale

Un emulatore di terminale che consente di programmare la porta seriale tramite script, in un linguaggio che ha tutti i costrutti fondamentali del basic, inclusa la sua proverbiale semplicità. Attraverso singole istruzioni, è possibile inviare comandi a dispositivi collegati alla seriale e, attraverso una semplice istruzione di assegnamento, è possibile immagazzinare il valore ritornato dal dispositivo in una variabile. Strutture decisionali, cicli e la possibilità di accedere al disco, definiscono la completezza dell'ambiente. L'utilizzo dell'applicazione è gratuito per trentuno giorni e per un massimo di cinquanta volte.

SerialBasic2_2.exe

SoftwareKey Trial Creator 1.10

Per creare versioni di prova del proprio software

Davvero semplice e ben fatto questo piccolo tool che consente di trasformare qualsiasi applicazioni Win32 in versione trial-30 giorni. Attraverso una procedura wizard, siamo guidati nelle varie scelte necessarie alla definizione del pacchetto di installazione. Gratuito.

TrialCreatorSetup.exe

AppMind 3.5.1

Migliora la qualità delle applicazioni in C e C++

Una soluzione completa per il controllo della qualità di applicazioni scritte in C e C++. Un insieme di tool che consente di migliorare al contempo prestazioni e robustezza attraverso tre fasi fondamentali: test, debug e tuning delle performance. Gratuito.

apm_win32_040506_0351BL777.exe

Come integrare nelle pagine Web una voce che guida l'utente

La pagina web "parla"

Da qualche tempo, un nuovo fenomeno ha timidamente iniziato a diffondersi online. Siti web che comunicano con i propri visitatori attraverso commenti vocali.

Non si tratta dei soliti screen reader, ma di veri e propri messaggi parlati pensati per semplificare la fruizione delle pagine e spesso associati a effetti visivi. Riflettendo sulla questione, potremmo chiederci come mai alcune web agency scelgano di dotare di audio le pagine di un sito, visto che gli screen reader sono completamente automatici. Per chi non ne avesse mai sentito parlare, gli screen reader - nati per venire incontro alle esigenze dei navigatori non vedenti, sono software in grado di leggere le pagine web create rispettando i parametri del W3C sull'usabilità. Nonostante la validità degli intenti, allo stato attuale diversi limiti tecnici rendono poco fluida la "lettura artificiale" del sito da parte dello screen reader. Pensiamo solo alla lingua: se il programma è ottimizzato per leggere l'inglese, difficilmente garantirà la corretta pronuncia di un testo in italiano. Inoltre, lo screen reader non riesce a dare la giusta enfasi alle parole: una stessa frase, pronunciata con un tono differente, può assumere diversi significati. L'alternativa è quindi creare commenti sonori letti da un bravo "speaker" e farli ascoltare al navigatore durante la connessione al sito. Il file *index.htm* presente nella cartella esempio_completo (disponibile nel CD-ROM allegato alla rivista) chiarisce quanto stiamo dicendo.

IL NOSTRO ESPERIMENTO

La pagina principale del sito web *http://www.ioprogrammo.it* è stata associata ad una presentazione audio costituita da voci che leggono alcune didascalie per descrivere le varie sezioni del sito dedicato alla rivista di programmazione più venduta in Italia. Le voci sono state registrate e convertite in file MP3, che vengono eseguiti secondo un ordine prestabilito. Il progetto è strutturato in modo da rendere facilmente aggiornabili i file audio e il loro ordine di esecuzione.

È importante sottolineare che non si tratta di una pagina diversa realizzata per l'occasione: è la home page disponibile online, a cui sono state apportate alcune piccole modifiche.

Quindi è lecito chiedersi: dov'è il trucco? Come vedremo tra breve, per ottenere questo risultato bastano un filmato Flash grande un pixel, un file XML dalla struttura molto semplice e qualche piccolo ritocco al file HTML.

METTIAMOCI ALL'OPERA

Prima di tutto, dobbiamo creare una sequenza di brevi messaggi vocali in formato MP3 e collocarli nella stessa cartella in cui si trova il file HTML a cui vogliamo associare l'audio. Sarebbe opportuno registrare i messaggi con una tonalità chiara e priva di inflessioni: se non vi sentite a vostro agio davanti a un microfono, potete coinvolgere un amico o un collaboratore che possa registrare i messaggi in modo adeguato. L'ideale sarebbe qualcuno che lavo-

Fig. 1: La prima pagina di ioProgrammo nella versione "audio", con didascalie interattive che guidano l'utente durante la navigazione.

ri in radio, faccia teatro o, in generale, sia abituato ad usare la propria voce in ambito professionale. Nell'esempio sul CD-ROM i file da eseguire in sequenza sono otto: *benvenuto1.mp3, primo.mp3, secondo.mp3, terzo.mp3, quarto.mp3, quinto.mp3, sesto.mp3* e *settimo.mp3*. Per essere precisi a questi andrebbe aggiunto anche *bentornato.mp3*, ma poiché costituisce un'eccezione, ne parleremo alla fine di questo articolo.

IL RUOLO DI XML

Gli otto file saranno caricati ed eseguiti in sequenza da un filmato Flash, nascosto nella pagina HTML. Flash, oltre a eseguire i file audio importati, può anche caricare uno alla volta in sequenza gli MP3 esterni. Il filmato che useremo contiene uno script che precarica ed esegue vari file audio seguendo un certo ordine. Tuttavia, se decidessimo di cambiare i nomi dei file, il loro numero o l'ordine di esecuzione, non saremmo costretti a modificare il sorgente *FLA*, poiché il filmato lavora in squadra con un file XML collocato nella stessa cartella.
Il file XML fornisce a Flash le seguenti informazioni:

- Il numero dei file MP3 da eseguire.
- L'ordine in cui devono essere eseguiti.
- I nomi dei file MP3.

Proprio questa caratteristica consente di aggiornare con facilità il nostro meccanismo, modificando solo i tag del file XML, senza dover intervenire continuamente sul filmato Flash. Di conseguenza, per riutilizzare il materiale disponibile sul CD-ROM della rivista, è sufficiente intervenire sul file XML e sostituire i vari file MP3 con i propri.
Dopo avere descritto a grandi linee il funzionamento, passiamo alla pratica. Nella stessa cartella che contiene l'esempio completo è disponibile anche la cartella tutorial. Al suo interno troviamo altre tre sottocartelle: *pagina_web, sorgenti_flash* e *MP3*.

- La cartella *pagina_web* contiene una versione ridotta della pagina principale del sito di ioProgrammo.
- La cartella *sorgenti_flash* contiene il file *esempio_base.fla*, il motore del nostro progetto.
- La cartella *MP3*, infine, contiene tutti i file audio necessari per la nostra sequenza.

Per associare l'audio alla prima pagina di *ioprogrammo.it*, prima di tutto salviamo la cartella *pagina_web* sul nostro hard disk. Preleviamo i file audio presenti nella cartella MP3 e collochiamoli nella cartella *pagina_web*. Analogo discorso vale per *esempiobase.fla* presente nella cartella *sorgenti_flash:* anche questo file dovrà essere posto nella cartella *pagina_web*.
Apriamo un editor di testo e scriviamo il seguente file XML:

```
<?xml version="1.0"?>
<brani>
<voce>benvenuto1.mp3</voce>
<voce>primo.mp3</voce>
<voce>secondo.mp3</voce>
<voce>terzo.mp3</voce>
<voce>quarto.mp3</voce>
<voce>quinto.mp3</voce>
<voce>sesto.mp3</voce>
<voce>settimo.mp3</voce>
</brani>
```

Chiamiamolo *listafile* e salviamolo con estensione *.xml* nella cartella *pagina_web*, scegliendo come tipo il semplice testo. Analizziamo in breve la struttura di questo semplicissimo file: è privo di riferimenti alla DTD e caratterizzato da un nodo principale costituito dall'elemento *<brani></brani>*. Il nodo principale prevede un certo numero di nodi figli (i tag *<voce></voce>*) che a loro volta contengono ognuno il nome di un file MP3.
Il contenuto dei vari tag viene definito anche nodo di testo e rappresenta uno dei due modi in cui possiamo inserire informazioni valide per Flash in un file XML; l'altro procedimento prevede l'uso degli attributi, cioè di coppie *nome="valore"* da inserire all'interno dei vari tag.
Nel nostro esempio, Flash legge i nodi di testo costituiti dai nomi dei file MP3, acquisendo tutte le informazioni che gli sono necessarie. Grazie ai tag XML, nel file Flash, per esempio, non sarà necessario specificare il numero di file da ascoltare.
Per adesso, adoperiamo questo tipo di struttura: successivamente modificheremo il file XML per aumentare il numero di dati da passare a Flash.

IL "FILMATO" FLASH

Apriamo il file *esempiobase.fla* con Flash MX 2004: per prima cosa notiamo che manca lo stage. Chi non ha mai adoperato Flash probabilmente non sa che al centro dell'interfaccia si trova un'area di lavoro (lo stage) rappresentata da un foglio bianco largo 550 e alto 400 pixel. Aprendo la finestra *Proprietà* del documento da *Elabora > Documento* (oppure con *CTR+J*), notiamo che la larghezza e l'altezza del documento sono state fissate a un pixel nei rispettivi campi di regolazione. Infatti, nel nostro progetto, non sono richieste le peculiarità grafiche di Flash,

Fin dalla versione 5, Flash è in grado di leggere documenti XML (Extensible Markup Language). I documenti XML si possono importare in Flash a patto di essere well formed, cioè "ben formati": questo significa che anche quando non fanno riferimento a specifiche DTD, devono rispettare tutti gli standard previsti. Allo stato attuale Flash non crea autonomamente i file XML, ma può solo leggerli. Ricorrere a file di questo tipo può costituire un metodo semplice e immediato per acquisire dati posti all'esterno di un filmato Flash. In altri termini, pur essendo a tutti gli effetti semplici file di testo, i file XML consentono di simulare una semplice base di dati grazie alle possibilità fornite dai vari nodi (i tag che li contraddistinguono).

ma la sua capacità di gestire l'audio. Quindi possiamo lavorare con un file Flash privo di contenuti grafici. Il codice si trova tutto nel primo fotogramma della Linea temporale. Per visualizzarlo, ricorriamo al percorso *Finestra > Pannelli di sviluppo > Azioni* (o, in alternativa premiamo il tasto *F9*). Tra breve analizzeremo il codice, ma per il momento occupiamoci di inserire il file Flash all'interno della home page di ioProgrammo.

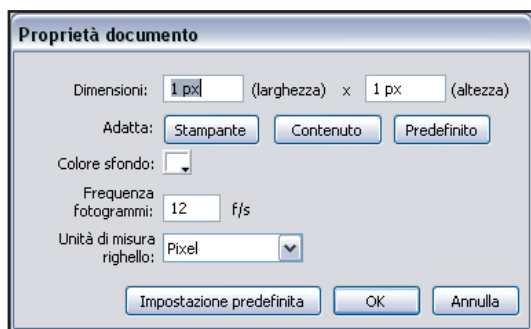



Fig. 2: **La finestra proprietà del documento consente di regolare le principali impostazioni del file Flash.**

Pubblichiamo il file, seguendo il percorso *File > Pubblica* (oppure premendo il tasto *F12*): nella stessa posizione in cui si trova il sorgente esempiobase.fla vengono creati anche i file *esempiobase.html* ed *esempiobase.swf.* Lanciando il file *esempiobase.html,* potremo già ascoltare le varie voci riprodotte in sequenza: il filmato SWF carica il file XML, che a sua volta indica il nome dei file MP3 da caricare.
A questo punto possiamo inserire il riferimento al filmato nella pagina di ioProgrammo. Apriamo *esempiobase.html* con un qualsiasi programma di videoscrittura e copiamo l'elemento *<object></object>* con tutto il suo contenuto. Apriamo con un programma di videoscrittura il file *index.htm*, che contiene la versione sintetica della prima pagina del sito di ioProgrammo, e incolliamo l'elemento *<object></object>* prima del tag *</body>*.
Con queste operazioni abbiamo collocato il file *swf,* che misura 1 pixel x 1 pixel, alla fine della pagina *index.htm,* senza correre il rischio di scompaginare (anche in modo impercettibile) il layout. Quando la pagina HTML si apre nella finestra del browser, viene riprodotta una sequenza di voci. Come accennato in precedenza, possiamo sostituire i commenti vocali inserendo il nome di altri file MP3 nel file XML, oppure aggiungere un file audio in più scrivendo un altro nodo *<voce></voce>*.

UNO SGUARDO AL CODICE

Riapriamo il file Flash e osserviamo il codice ActionScript 2.0. Tutto il meccanismo dello script si basa sul caricamento del file XML:

```
var archivio:XML= new XML;
archivio.ignoreWhite=true;
archivio.onLoad=function():Void{
  var elenco:Array = new Array();
  for(var a:Number = 0; a <
                this.firstChild.childNodes.length; ++a){
    elenco[a]= this.firstChild.childNodes[a]
                              .firstChild.nodeValue;
  }
  // codice relativo al suono
}
archivio.load("listafile.xml");
```

Flash crea un nuovo oggetto XML denominato *archivio*. La proprietà *ignoreWhite* accetta solo valori di tipo booleano e per impostazione predefinita ha come valore *false.*
Attribuendo a questa proprietà il valore *true,* si fa in modo che gli spazi vuoti siano ignorati e si evita di considerare come nodi di testo anche le semplici tabulazioni. Attraverso il gestore di eventi *onLoad* vengono eseguite una serie di istruzioni in seguito al caricamento del file XML e alla successiva analisi dello stesso file nell'oggetto archivio. Prima di procedere oltre, riepiloghiamo in breve alcune delle proprietà che consentono a Flash di analizzare un file XML.

- **firstChild** consente di individuare il nodo principale di un file XML. Scrivendo *nomeoggetto .firstChild,* oppure con la sintassi relativa *this. fisrtChild,* possiamo accedere al nodo principale (che nell'esempio è *<brani>*).

- **chlidNodes[i]** individua i vari nodi figli che compongono il file XML, attraverso una struttura simile a quella degli Array.

- **NodeValue** restituisce il valore di un nodo di testo.

Grazie a queste proprietà, adoperando la sintassi del punto, possiamo leggere tutte le informazioni di un file XML. Per esempio scrivere *this.firstChild.child Nodes.length;* ci permette di sapere quanti nodi figli ci sono all'interno del documento.
Su questa base possiamo comprendere meglio la finalità del ciclo *for* che, dopo avere contato tutti i nodi del documento XML, passa all'array elenco tutti i nodi di testo. Di conseguenza:

- scrivere *this.firstChild.childNodes[0].firstChild. nodeValue* equivale a scrivere *benvenuto1.mp3*;

- scrivere *this.firstChild.childNodes[1].firstChild. nodeValue* equivale a scrivere *primo.mp3*;

- scrivere *this.firstChild.childNodes[2].firstChild.*

nodeValue equivale a scrivere *secondo.mp3;*
e così via

Il nostro obiettivo è quello creare un Array elenco, in cui si trovino tutti i file MP3. Per individuare i file da caricare ed eseguire, ActionScript farà riferimento a questo specifico Array.

```
index.htm - Blocco note
File  Modifica  Formato  Visualizza  ?
            <!-- tag filmato Flash -->
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
codebase="http://fpdownload.macromedia.com/pub/shockwave/ca
bs/flash/swflash.cab#version=7,0,0,0" width="1" height="1"
id="audio" align="middle">
<param name="allowScriptAccess" value="sameDomain" />
<param name="movie" value="audio.swf" />
<param name="quality" value="high" />
<param name="bgcolor" value="#ffffff" />
<embed src="audio.swf" quality="high" bgcolor="#ffffff" width="1"
height="1" name="audio" align="middle"
allowScriptAccess="sameDomain"
type="application/x-shockwave-flash"
pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
            <!-- fine tag filmato Flash -->
</BODY></HTML>
```

Fig. 3: ***Nel pannello Azioni visualizziamo il codice relativo alla gestione dei file MP3.***

Nell'ultima riga di codice, il metodo *load()* specifica quale file XML deve essere caricato. Anche se può sembrare una contraddizione, per assicurare il corretto funzionamento del codice è importante utilizzare il metodo *load()* solo dopo avere specificato le istruzioni del gestore *onLoad()*. In questo modo si dà il tempo al Flash Player di acquisire le operazioni da compiere nel momento in cui il file XML è stato caricato. Del resto, considerando che il nostro file XML pesa un solo Kb, possiamo facilmente immaginare con quale velocità Flash carichi il file.

NOTA

I file di esempio disponibili sul CD hanno una valenza puramente didattica, per cui la loro dimensione (in alcuni casi eccessiva) non deve essere presa come punto di riferimento: quando si preparano i messaggi parlati, bisogna evitare che uno o più MP3 abbiano una durata (e quindi un numero di Kb) superiori al messaggio iniziale. Questo squilibrio potrebbe creare rallentamenti al momento dell'accesso alla pagina web, in presenza di una connessione lenta. L'ideale sarebbe creare un file iniziale un po' più grande, con tutti gli altri più piccoli e possibilmente accomunati da una dimensione simile. Con questo sistema, eseguito il primo, gli altri hanno il tempo di essere caricati e, possedendo dimensioni simili, non causano intervalli troppo lunghi tra un'esecuzione e l'altra.

SCEGLIERE IL RITMO

Il codice descritto finora riguarda solo l'interazione con il file XML. Abbiamo tralasciato al codice relativo alla gestione del suono che si trova prima dell'ultima parentesi graffa. Aprendo il file Flash notiamo un ulteriore blocco di codice di codice, inserito al posto del commento riportato nello script precedente. Prima di soffermarci sul funzionamento di queste righe di codice, chiariamo quali sono le nostre esigenze. Ci sono un paio di importanti aspetti tecnici con cui fare i conti.

1. Non è opportuno caricare tutti insieme i file MP3 prima di eseguirli, perché si rischia di produrre notevoli tempi di attesa dovuti al caricamento che precede l'esecuzione dei messaggi.

2. Non conviene caricare ed eseguire i file MP3 uno alla volta: i navigatori dotati di connessioni lente potrebbero riscontrare fastidiose pause tra un messaggio vocale e l'altro.

L'unica possibilità per ottimizzare i tempi, è fare in modo che non appena un file viene caricato inizi subito il caricamento del file successivo. Allo stesso tempo, bisogna controllare ad ogni istante, quali file sono stati caricati in modo da eseguirli appena pronti. Questo meccanismo ci consente di eseguire i suoni mentre altri vengono contemporaneamente caricati - una soluzione inusuale rispetto alla "classica" procedura di caricamento dei suoni esterni.
Lo script per caricare un file MP3 in genere ha una sintassi del tipo:

```
var miosuono:Sound=new Sound();
miosuono.onLoad = function (caricato:Boolean)
{
  if(caricato)
  {
    miosuono.start();
  }
}
miosuono.loadSound("primo.mp3",false);
```

In breve, viene creato un nuovo oggetto *Sound* chiamato *miosuono* e, quando Flash tenta di caricare il suono con il gestore di evento *onLoad*, viene accertato il valore *true* della variabile *caricato*, che è tale solo se il file è stato importato correttamente.
Il metodo *start()*, in grado di attivare il suono, entra in azione solo se la condizione *caricato* è soddisfatta. Nell'ultima riga di codice, come nel caso del file XML, si ricorre al metodo *loadSound()* per caricare il file. Il metodo in questione prevede due parametri: il primo specifica il nome ed il percorso del file MP3, il secondo il tipo di caricamento, che prevede solo due valori, *true* (in streaming) e *false* (senza streaming). Se vogliamo evitare interruzioni durante l'esecuzione del suono, dobbiamo inserire *false* come secondo parametro. Se inserissimo questo script all'interno di un fotogramma chiave in un file Flash e pubblicassimo il file in una cartella contenente un file denominato *primo.mp3*, otterremmo la riproduzione del suono. Osservando il codice, verifichiamo che i due aspetti del meccanismo (caricamento ed esecuzione) sono fortemente correlati tra loro, per que-

sto, nel nostro caso è necessario separarli. Il codice che si trova nel primo fotogramma del filmato *esempiobase.fla* è caratterizzato da due funzioni ricorsive: *carica()* ed *esegui()*. In particolare, la funzione *carica()*, oltre ad applicare in sequenza il metodo *loadSound()* per tutti i file MP3 segnalati dal documento XML, crea un Array chiamato *caricati* a cui aggiunge di volta in volta nuovi elementi tramite il metodo *push()*. La funzione *esegui()* è strutturata in modo tale da verificare continuamente se esiste un nuovo elemento nell'array caricati. Non appena viene segnalata la presenza di un nuovo file caricato, si applica il metodo *start()* per quello specifico suono.

IL MESSAGGIO DI BENTORNATO

Un messaggio di questo tipo serve ad accogliere un utente che ha già visitato la nostra pagina web: per realizzarlo dobbiamo ricorrere ad un cookie creato direttamente con Flash.
A partire dalla versione MX infatti, Flash consente di depositare file con estensione *.sol* sulla macchina dell'utente. Con questo metodo, ogni volta che viene attivato il filmato SWF, possiamo verificare se l'utente possiede il cookie sul suo hard disk. Se il cookie è assente, viene caricato il file *listafile.xml* che innesca la sequenza di MP3 a partire dal messaggio *benvenuto1.mp3;* in caso contrario viene caricato un altro file XML chiamato bentornato.xml, che innesca la sequenza di MP3 a partire dal messaggio *bentornato.mp3*. Creiamo il file *bentornato.xml* scrivendo gli elementi illustrati nella Fig. 4 e salviamolo nella solita cartella *pagina_web*.
A questo punto, dobbiamo solo aggiungere il codice necessario alla creazione del cookie. Il codice in questione, da inserire al posto dell'istruzione *archivio.load("listafile.xml")*, l'ultima dello script contenuto in *esempiobase.fla*, è il seguente:

```
shared=SharedObject.getLocal("vis");
confronta=shared.data.vl;
if(confronta=="visto")
{
   archivio.load("bentornato.xml");
}
else
{
   archivio.load("listafile.xml");
   // crea il cookie
   shared=SharedObject.getLocal("vis");
   shared.data.vl="visto";
   shared.flush();
}
```

Lo script, prima di tutto, si accerta della presenza del cookie. Essendo la prima volta che l'utente visita la pagina, sul suo hard disk non è stata precedentemente depositata alcuna informazione, per cui la condizione *if()* non è vera. In questo caso vengono eseguite le istruzioni associate alla condizione else, che caricano il documento *listafile.xml*. Nelle ultime tre righe di codice viene creato il cookie: al suo interno la stringa "visto" è assegnata alla proprietà vis. La presenza di questa stringa ci consente di verificare se il cookie è stato depositato: nel caso in cui, recuperando il valore di *vis* attraverso la variabile confronta, otteniamo *"visto"*, viene caricato il file *bentornato.xml*.

CONCLUSIONI

Osservando il file *index.htm* presente nella cartella *esempio_completo*, noterete, oltre al commento sonoro, che la pagina web è associata a finestre DHTML che si aprono vicino ai link descritti. Questo effetto si basa sul semplice utilizzo dei cosiddetti layer (conosciuti in italiano anche come livelli): si tratta di tag *<DIV>-</DIV>* contraddistinti da una serie di attributi attraverso i quali è possibile collocare elementi HTML su livelli sovrapposti. I layer possono essere resi invisibili o mantenuti visibili, sia attraverso comuni attributi HTML, sia dinamicamente tramite JavaScript. Con Dreamweaver MX 2004 sono stati creati otto layer (uno per ogni messaggio), impostati come invisibili. È stata aggiunta anche un breve funzione JavaScript in grado di renderli visibili a certe condizioni. Poiché le didascalie devono essere sincronizzate con i file audio, è stato necessario innescare la funzione JavaScript tramite Flash. Anche in questo caso, per rendere il tutto facilmente aggiornabile, a dettare l'ordine di apparizione delle finestre DHTML è il file XML importato da Flash. Poiché lo spazio è tiranno, affronteremo questo argomento sul prossimo numero. Per adesso, se volete iniziare a dare un'occhiata al meccanismo, nella cartella *esempio_completo* sono disponibili i file XML e HTML, mentre nella cartella *tutorial/sorgenti_flash* si trova il file *audio.fla*

Maurizio Battista

```
28 function carica(i){
29 suono[i].onLoad = function(conferma:Boolean):Void {
30 if (conferma) {
31 caricati.push(elenco[i]);
32 ++i;
33 carica(i);
34 }
35 }
36 suono[i].loadSound(elenco[i], false);
37 }
38 // funzione che esegue i suoni in sequenza
39 function esegui(s:Number):Void{
40 _root.onEnterFrame=function(){
41 if(caricati[s]!=undefined){
42 suono[s].start();
43 delete _root.onEnterFrame;
44 }
45 setInterval(ciclo,500);
46 function ciclo():Void{
47 suono[s].onSoundComplete=function(){
48 p+=1;
49 esegui(s+=1);
50 clearInterval(c);
51 }
52 }
53 }
54 }
55 esegui(0);
56 }
57 archivio.load("listafile.xml");
```

Fig. 4: ***Il file bentornato.xml, presenta una diversa sequenza di MP3 da caricare ed eseguire.***

J2ME Provisioning in MIDP 2.0

Eseguire i MIDlet sui cellulari

Non è gratificante sviluppare MIDlet se poi non possiamo vederli in esecuzione. Daremo uno sguardo alle varie possibilità per far "girare" le nostre applicazioni J2ME sui nostri cellulari.

Nell'articolo del numero scorso di ioProgrammo dedicato al J2ME, dopo aver creato un'applicazione che metteva insieme nozioni legate allo sviluppo di GUI e ai protocolli di rete sotto MIDP e CLDC, ci siamo lasciati con la promessa di vedere come i MIDlet venissero installati sui nostri cellulari per poterli ammirare finalmente davvero in azione. Per chi si fosse perso la rivista del mese passato, in questo numero c'è la recensione di EclipseME nella quale potete trovare una (seppur molto breve) introduzione a quello che J2ME rappresenta. Ovviamente, per poter portare un'applicazione sul nostro dispositivo mobile, dobbiamo innanzitutto avere l'applicazione! Al fine di non distrarre l'attenzione dall'obiettivo primario della nostra discussione, utilizzeremo un MIDlet molto semplice, praticamente lo stesso creato nella recensione di EclipseME menzionata poc'anzi, con l'aggiunta solo di un Ticker, giusto per poter vedere qualcosa che si muove.

IL MIDLET

Partirò dal presupposto che stiate utilizzando Eclipse per lo sviluppo Java insieme al plug-in EclipseME per la creazione di progetti wireless e che abbiate installato il *Sun Wireless Toolkit 2.1* come run-time J2ME. In realtà, l'unica cosa importante è che alla fine di questa sezione abbiate una MIDlet Suite impacchettata in un JAR e con relativo file di descrizione JAD. Utilizzando i wizard di EclipseME, creiamo quindi il nostro progetto *MIDlet Suite* e configuriamolo per MIDP 1.0, onde evitare problemi di compatibilità di versione con il cellulare: anche se siete sicuri che MIDP 2.0 sia supportato, potrebbe non essere però supportato CLDC 1.1, e bisognerebbe mettersi a modificare il JAD (niente di trascendentale, ma per ora vogliamo che fili tutto super-liscio!). Sempre con i wizard, creiamo adesso un nuovo MIDlet dentro il progetto appena generato, dandogli, per esempio, il nome di *ProvisioningMidlet* (il perché della mia scelta sarà chiaro un po' più avanti) nel package *mymidlets* ed implementando l'interfaccia *CommandListener* di MIDP. Il codice completo del MIDlet lo trovate nel *Listato 1*, con i commenti eliminati per questioni di spazio. Come potete notare, si tratta di un'applicazione J2ME molto semplice. A livello di istanza della classe definiamo la schermata principale *(scrMain)*, un ticker (*tkrBanner*, si tratta di una scritta che scorre, normalmente in alto sul display del cellulare) ed un comando *(cmdExit)*. La schermata è una semplicissima TextBox, a cui aggiungiamo il ticker *tkrBanner* e che avrà infine come unico comando il nostro *cmdExit* che utilizzeremo per permettere all'utente di uscire dall'applicazione. La gestione del comando avverrà ad opera del MIDlet stesso nel metodo di callback *commandAction*, obbligatorio dato che abbiamo scelto di implementare *CommandListener.*
Dei metodi del ciclo vitale del MIDlet *(startApp, pauseApp,* e *destroyApp)*, l'unico che necessita di implementazione nel nostro caso è *startApp*, dove impostiamo il nostro screen *scrMain* come display corrente dell'applicazione dopo avergli aggiunto *ticker* e *command*. Infine, sempre nello stesso metodo, dichiariamo che i comandi di *scrMain* devono essere gestiti dal MIDlet ad opera della chiamata *setCommandListener.* A mo' di ripasso, e per venire incontro a chi si avvicina per la prima volta al J2ME, ricordiamo solo alcuni punti fondamentali relativi alle applicazioni per dispositivi mobili. Il MIDlet rappresenta l'applicazione J2ME ed ha tre stati vitali: non-esiste, attiva, in-pausa. Quando chiediamo al nostro cellulare di eseguire un programma Java, questo istanzia il MIDlet (invocandone così il costruttore, che funge da inizializzatore dell'applicazione) ed invoca il metodo *startApp.* Se mentre state utilizzando

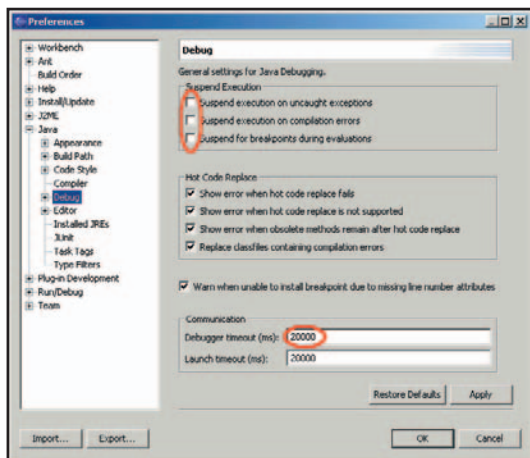



Fig. 1: ***La configurazione del debugger di Eclipse per EclipseME.***

il software qualcosa ne interrompe l'esecuzione (per esempio, vi squilla il cellulare o vi arriva un SMS), il codice verrà messo in pausa a seguito di una chiamata a *pauseApp*, dove il programmatore può eventualmente salvare lo stato dell'applicazione. Alla fine della vostra conversazione telefonica o dopo che avete letto il messaggio di testo, il cellulare rimette in moto il MIDlet, invocando di nuovo *startApp:* siete voi programmatori a dover sapere in qualche modo che l'applicazione era già stata fatta partire prima. Infine, per eliminare del tutto l'applicazione dalla sua memoria, il cellulare distruggerà il *MIDlet*, invocando il metodo *destroyApp* prima di eliminare l'oggetto. Tenete sempre a mente che i tre metodi visti appena adesso, come già spiegato nell'articolo del mese scorso, sono delle notifiche da parte del sistema verso la nostra classe: se nel codice del MIDlet ho necessità di mettere in pausa l'applicazione o distruggerla, non ha senso chiamare *pauseApp* o *destroyApp* solamente, devo richiedere l'operazione al sistema con *notifyPaused* o *notifyDestroyed*. In maniera del tutto simile, se voglio far ripartire il MIDlet in pausa (ad esempio da un thread parallelo in esecuzione), invocherò *resumeRequest*, ma in questo caso *startApp* sarà invocato automaticamente dal sistema quando l'applicazione viene rimessa in moto (*pauseApp* e *destroyApp* devono invece essere invocati esplicitamente dal programmatore). Relativamente alle interfacce grafiche, ricordiamo infine che i dispositivi compatibili con il profilo MIDP mettono a disposizione il loro display per la visualizzazione di uno screen (schermata) alla volta da parte delle applicazioni. Un riferimento al display del cellulare lo si ottiene con il metodo statico *getDisplay* di *Display*, mentre per ottenere o impostare lo screen corrente si usano *getCurrent* e *setCurrent* dello stesso oggetto. Un'applicazione J2ME complessa svolge buona parte del suo lavoro modificando lo screen corrente per offrire le varie parti della propria interfaccia grafica. Ogni istanza di uno screen può poi definire un proprio menu di comandi tramite una serie di oggetti *Command* aggiunti ad opera del metodo *addCommand*: sarà il cellulare a stabilire dove questo menu sarà visualizzato in base alla propria grafica, mentre per ricevere la scelta dell'utente è necessario implementare *CommandListener* per avere il metodo di notifica *commandAction*, che vi passa il comando selezionato e lo screen attivo.
A questo punto, testiamo rapidamente la nostra applicazione per essere sicuri che funzioni sull'emulatore prima di portarla sul cellulare. Accertatevi, innanzitutto, di aver configurato correttamente i parametri del debugger Java per poterlo usare con EclipseME (vedi Fig. 1). Selezionate il MIDlet *ProvisioningMidlet* nella view del *Package Explorer* e selezio-

```
package mymidlets;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
public class ProvisioningMidlet extends MIDlet
                              implements CommandListener {
  private Command cmdExit = new Command("Esci
                          dal MIDlet", Command.EXIT, 0);
  private Ticker tkrBanner = new Ticker("
    ioProgrammo - Federico Mestrone - J2ME Tutorial");
  private Screen scrMain = new TextBox("Welcome",
   "Questo è il mio primo MIDlet!", 100, TextField.ANY);
  public ProvisioningMidlet() {
    super();}
  protected void startApp() throws
                          MIDletStateChangeException {
    Display d = Display.getDisplay(this);
    if (d.getCurrent() == null) {
      scrMain.setTicker(tkrBanner);
      scrMain.addCommand(cmdExit);
      scrMain.setCommandListener(this);
      d.setCurrent(scrMain);}}
  protected void pauseApp() {}
  protected void destroyApp(boolean arg0) throws
                          MIDletStateChangeException {}
  public void commandAction(Command arg0,
                               Displayable arg1) {
    if (arg0 == cmdExit) {
      try { destroyApp(false);
              notifyDestroyed();}
      catch (MIDletStateChangeException msce) {}
} } }
```

LISTATO 1: Il MIDlet da portare sui nostri cellulari

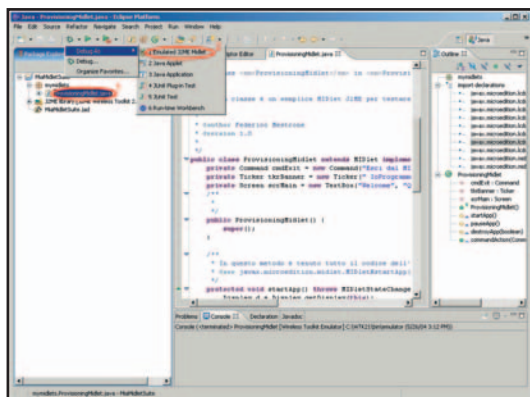



Fig. 2: ***Esecuzione in debug di un MIDlet sotto l'emulatore con EclipseME.***

Fig. 4: ***Il nostro MIDlet gira su un cellulare virtuale.***

Fig. 3: ***Il display dell'emulatore con lo screen del nostro MIDlet.***

nate il menu *Run->Debug As->Emulated* J2ME Midlet, oppure utilizzate la toolbar come in Fig. 2. Dopo alcuni istanti dovreste vedere l'emulatore del *Sun Wireless Toolkit* aprirsi e a seguito di una breve pausa la vostra applicazione appare sul display del cellulare virtuale (vedi Fig. 4). Nella Fig. 3 abbiamo uno zoom sul display del dispositivo con il nostro screen come corrente. Riconoscerete nella figura il *ticker* (contrassegnato dal cerchio giallo 1), il titolo dello screen impostato nel costruttore della TextBox (cerchio 2) e il nostro unico comando (cerchio 3) che ha preso il posto normalmente lasciato al menu contestuale dell'apparecchio.
Il resto dello screen è occupato dalla casella di testo.

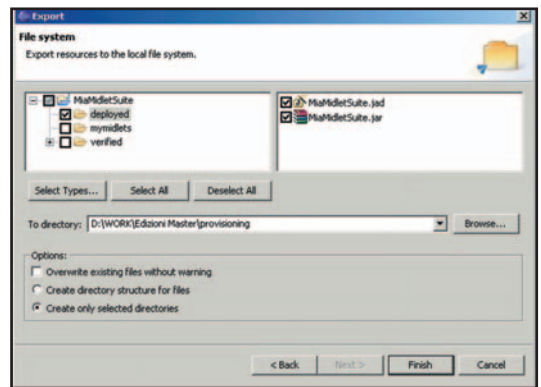
KVM, CLDC E MIDP

Il J2ME per i piccoli dispositivi mobili (cellulari e PDA) definisce una configurazione di base (detta CLDC) che include la definizione di una macchina virtuale alleggerita (la KVM) e una libreria per l'accesso alle funzionalità essenziali dell'hardware (I/O, gestione della memoria, etc). Ad un livello superiore, il profilo (MIDP) prevede una libreria rivolta ai programmatori J2ME per lo sviluppo di applicazioni e la creazione delle interfacce grafiche.

LA MIDLET SUITE

Adesso non ci resta altro che prepararci per installare l'applicazione Java sul nostro cellulare compatibile J2ME. Per prima cosa chiediamo ad EclipseME di organizzare tutto il materiale necessario sparso per il progetto Eclipse in una *MIDlet Suite standard*. Questa operazione si concretizzerà nella generazione di un'accoppiata *JAR* + *JAD* pronta per il provisioning, ovvero sia la distribuzione su dispositivo J2ME (questo termine tecnico ha dato anche il nome alla nostra classe *MIDlet,* se ricordate!). La procedura è molto semplice: click col destro sul progetto, menu *J2ME->Create* Package (come in Fig. 5). Nella cartella *deployed* (invisibile nel *Package Explorer* a causa di un filtro) avrete ciò che serve per il *provisioning.*
A questo punto esportiamo da Eclipse i due file che ci servono: menu *File->Export…*, esportate come *File System*, poi selezionate solo la cartella *deployed* del vostro progetto *MIDlet Suite* e scegliete un percorso di destinazione a voi comodo dove mettere il tutto: la schermata di export la trovate in Fig. 6. Da adesso, quando parlo della nostra *MIDlet Suite* mi riferisco ai due file che stiamo esportando in questo momento, quindi tenete bene a mente dove li state mettendo!

Fig. 5: **Preparare la MIDlet Suite con il menu di EclipseME.**

IL PROVISIONING

Giunti fino qui, ora dobbiamo scegliere come portare il nostro codice sul cellulare per poterlo mettere in esecuzione. Nel numero scorso avevamo accennato all'AMS (*Application Management Software,* detto anche *Java Application Manager* o JAM). Questo programma si occupa dell'installazione delle applicazioni J2ME sui dispositivi MIDP ed è proprio dell'AMS che avremo bisogno per il *provisioning.*
Esso infatti viene tirato in ballo ogniqualvolta abbiamo a che fare con JAD e JAR. Abbiamo due opzioni davanti a noi: la prima richiede che si stabilisca una connessione tra il PC ed il telefonino, la seconda invece può prescindere da questa comunicazione, ma non è detto che sia supportata dal vostro modello di cellulare. Se optate per la prima possibilità, ci sono normalmente tre varianti per mettere in contatto il computer con il telefono: l'infrarosso, Bluetooth, oppure un cavo seriale che si connette al cellulare. È chiaro che qui ci stiamo addentrando in territorio minato, perché la soluzione ideale in questo caso dipende da molti fattori, non ultimo dei quali le capacità dei nostri hardware (mobili e non). Di fatto queste tecnologie non prevedono altro che il trasferimento di un file dal PC al telefono, e l'utilizzo esplicito dell'AMS da parte dell'utente per installare e mettere in esecuzione la Suite scaricata. La modalità di trasferimento delle risorse va valutata di volta in volta consultando le specifiche del driver e del software che state utilizzando ed è quindi difficile dare delle direzioni generiche in questo senso. La seconda possibilità, invece, offre un approccio che non richiede di connettere PC e cellulare ed è decisamente più generica della prima. Si tratta dell'OTA *(Over The Air) provisioning* ed è una modalità divenuta obbligatoria per i dispostivi J2ME a partire dalla versione 2 del profilo MIDP, anche se è comunque spesso supportata da apparecchi con MIDP 1. In base a questo protocollo di distribuzione, il programmatore mette il proprio lavoro (nel senso di JAR e JAD!) sotto la document root di un web server di modo che sia possibile scaricare la *MIDlet Suite* da browser. Dall'altro lato, il dispositivo wireless effettua una connessione a Internet ed accede all'indirizzo a cui è raggiungibile il file JAD di descrizione. Riconoscendo la tipologia di contenuto scaricato, il cellulare passerà il file *.jad* all'AMS, il quale – grazie alla voce *MIDlet-Jar-URL* del file *.jad* stesso (configurata

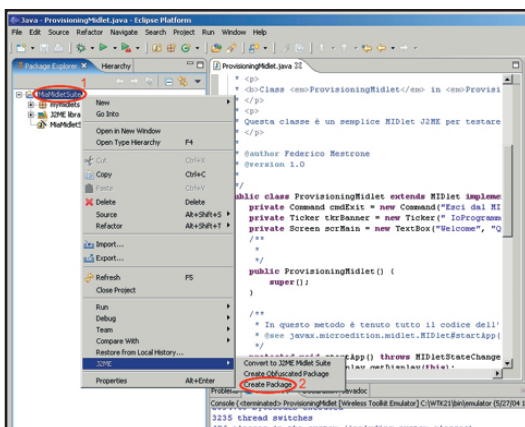



Fig. 6: **Esportazione della MIDlet Suite (JAR e JAD) da Eclipse.**

automaticamente da EclipseME se lo usate) – sarà in grado di andarsi a scaricare anche il file *.jar* e di dare inizio alla procedura di installazione, chiedendo le dovute conferme al proprietario. L'unico punto in cui è richiesta un po' di manipolazione è la configurazione del web server: a quel livello, infatti, è necessario che venga inviata un'intestazione HTTP corretta quando sono richiesti i file *.jad* e *.jar* affinché il *Content-Type* che arriva al cellulare sia rispettivamente un tipo MIME *text/vnd.sun.j2me.app-descriptor* e *application/java*. Questa semplice associazione tra un'estensione di file ed una stringa di tipo MIME ovviamente dipende dal web server che state usando.

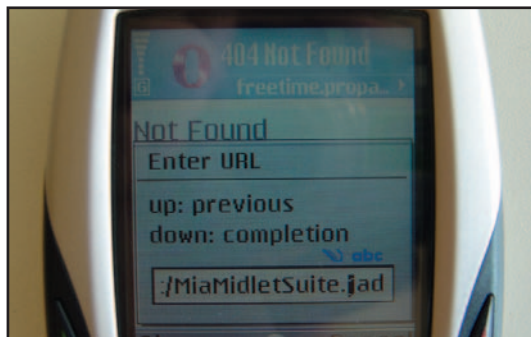



Fig. 7: ***OTA Provisioning: Immetto sul browser l'indirizzo del file .jad.***

L'APPLICAZIONE SUL TELEFONINO

Analizziamo adesso la nostra suite. Il mio web server è Apache, per cui prendo il file *httpd.conf* ed aggiungo le seguenti due linee:

```
AddType text/vnd.sun.j2me.app-descriptor .jad
AddType application/java .jar
```

che fanno sì che a fronte di un .jad o .jar il server dia come contenuto MIME la stringa specificata. A questo punto faccio l'upload di *MiaMidletSuite.jad* e *MiaMidletSuite.jar* (quelli che avete esportato da

Fig. 8: ***OTA Provisioning: Il cellulare riconosce che il file è per l'AMS.***

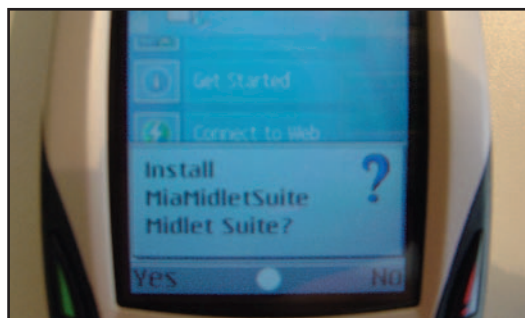



Fig. 9: ***OTA Provisioning: L'AMS installa la Suite previa conferma.***

Eclipse o se volete quelli dal CD allegato) sulla document root del mio host tramite ftp (la mia cartella è */var/www*, ma questo dipende dall'installazione di Apache). Adesso verifico dal mio PC che i file siano effettivamente raggiungibili via internet, navigando all'indirizzo *http://<nome del vostro host>/MiaMidletSuite.jad*, e ovviamente cancello quando il browser mi propone di scaricare. Faccio la stessa prova con *MiaMidletSuite.jar* e, se anche quella è raggiungibile, sono pronto a passare al cellulare. Sul mio Nokia 6600, apro il browser *Opera* (dovete avere già configurato il gateway per l'accesso Internet via GPRS: chiedete al vostro operatore di telefonia, ci pensano loro) ed immetto l'URL del file JAD (Fig. 7). Appena il server risponde, viene invocato l'AMS per prendere in carico il file di tipo *text/vnd.sun.j2me.app-descriptor* (Fig. 8), e l'AMS provvede a scaricare il JAR e ad installarlo previa conferma (Fig. 9).

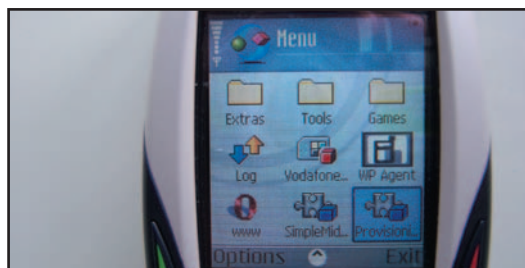



Fig. 10: ***I MIDlet della Suite fanno ora parte del menu del telefonino.***

Ora nel menu del mio telefonino ho a disposizione una voce nuova (Fig. 10), e se provo a selezionarla… mi ritrovo (Fig. 11) la mia prima applicazione MIDlet!

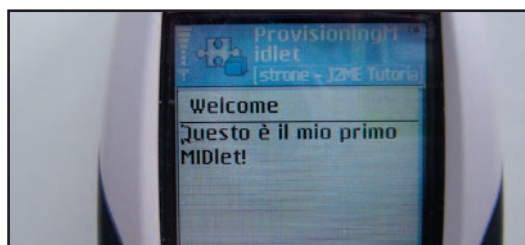



Fig. 11: ***Il MIDlet in esecuzione su un Nokia 6600.***

Federico Mestrone

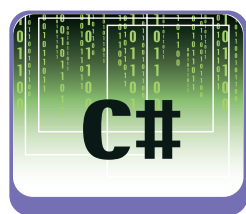
MIDLET E MIDLET SUITE

Il MIDlet è la classe Java che incorpora un'applicazione J2ME, implementandone le varie fasi del ciclo vitale e fornendo il punto di partenza per l'interazione con l'utente. La *MIDlet Suite* invece è l'insieme di un file JAR dove sono raccolti uno o più MIDlet, unito ad un descrittore delle applicazioni contenute che è il file JAD.

Sul sito OnJava, all'indirizzo www.onjava.com/topics/java/Wireless_Java trovate un'intera sezione dedicata al wireless e allo sviluppo di MIDlet. È un ottimo punto di partenza per approfondire le vostre conoscenze e trarre ispirazione per le vostre applicazioni J2ME. Nel frattempo, se vi sorge qualche dubbio sul materiale di questo articolo, potete sempre contattarmi all'email federico.mestrone@ioprogrammo.it!

Automatizzare la gestione di magazzino con tag a radiofrequenza

Un gestionale a radiofrequenza

La gestione di un magazzino è per molte aziende una delle parti cruciali dell'attività. Vediamo come le tecnologie attuali ci aiutano a semplificare tutto ciò, velocizzando il nostro lavoro.

La gestione informatizzata di un magazzino ha come scopo principale quello di rispondere a domande tipo: quanta merce ho in magazzino? Ne ho abbastanza? Quanto vale il mio magazzino oggi? Dove si trova il prodotto *xyz*? Per rispondere a queste domande, numerose software house hanno realizzato decine di prodotti accomunati da una logica semplice ed abbastanza efficace: si inseriscono le quantità della merce arrivata, si tolgono quelle della merce in uscita e si ottiene il totale magazzino.
Questo dovrebbe bastare a garantire l'integrità dei dati ma, chiunque ha dovuto gestire un magazzino, sa che purtroppo non è sempre così e periodicamente va fatto un inventario. Il tempo necessario ad effettuare l'inventario dipende da tre fattori fondamentali:

- la quantità di prodotti presenti
- il numero di persone coinvolte
- il sistema utilizzato per fare l'inventario

Inutile precisare che maggiore sarà il tempo impiegato, maggiore sarà il costo dell'operazione! Essendo programmatori e non potendo agire sui primi due punti, vedremo come agire sul terzo sfruttando nuove tecnologie sia software che hardware, per velocizzare le operazioni comuni.
Alla fine di questi articoli avremo un software che ci consentirà di gestire un magazzino in tempi rapidi e fare un inventario senza muovere i prodotti dagli scaffali. Entreremo nel futuro degli RFID, la tecnologia ancora in fase di sperimentazione che promette di rivoluzionare il settore della logistica e vedremo come usarla nei nostri software. Che dobbiate occuparvi di un negozio di abbigliamento, di un rivenditore di pezzi di ricambio o di un grande magazzino, avrete a disposizione una base da cui partire. Iniziamo…

I TAG

Uno dei componenti di maggiore importanza nei sistemi di gestione magazzino è l'etichetta: siamo ormai abituati a vedere un codice a barre su qualsiasi prodotto in commercio. Sebbene questo sistema sia ormai enormemente diffuso, non è esente da problemi e limiti tecnologici.

Fig. 1: ***Un tag a radiofrequenza.***

In primo luogo, l'etichetta può facilmente deteriorasi rendendo il codice illeggibile. In secondo luogo, l'utilizzo dei codici a barre è sconsigliato in ambienti polverosi. Basta infatti una piccola macchia sul codice per renderlo illeggibile. Queste problematiche sono tutte legate ad una caratteristica intrinseca del sistema: il lettore deve poter "vedere" il codice, pena la non leggibilità. La soluzione a questi problemi arriva con l'avvento dei tag (o transponder) a radiofrequenza. Il loro scopo è fondamentalmente quello di trasmettere il codice univoco memorizzato al loro interno ad un apposito lettore, attraverso una trasmissione radio che lavora su una frequenza ben definita. I tag a radiofrequenza si distinguono in due famiglie principali: *attivi* e *passivi.*

Quelli *attivi* possono trasmettere autonomamente il loro codice che può essere letto da una distanza di 15/20 metri, il che li rende particolarmente adatti in alcuni campi applicativi come, ad esempio, il Telepass autostradale. Lo svantaggio è che devono essere alimentati da una fonte esterna, cosa che li rende ingombranti e costosi.

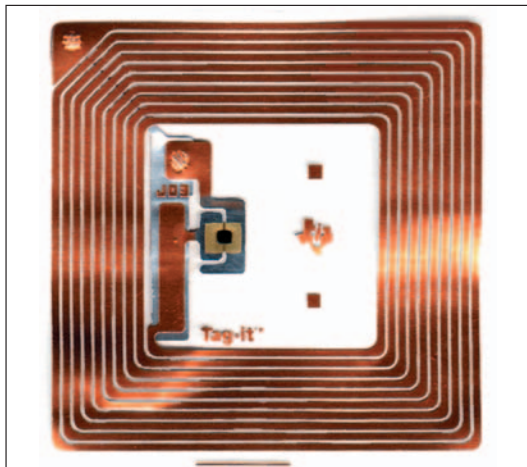



Fig. 2: ***Il dettaglio di un tag in cui è chiaramente visibile l'antenna e la parte attiva.***

I tag *passivi* sono invece piccoli, leggeri ed economici. Sono costituiti da una memoria non volatile (in cui è memorizzato un codice univoco), un circuito di trasmissione ed una antenna il cui scopo è di alimentare memoria e circuito di trasmissione. Il principio di funzionamento di questi tag è molto semplice: il lettore genera un campo elettromagnetico che viene captato dall'antenna del tag. Tale campo genera, per induzione, una tensione elettrica che carica un piccolo condensatore. Quando la tensione ai capi del condensatore raggiunge un determinato valore, il circuito di trasmissione del tag trasmette il codice contenuto nella memoria non volatile. Tutto il procedimento avviene in poche frazioni di secondo rendendo possibile la lettura di molti tag in brevissimo tempo. La distanza di utilizzo però, a differenza dai tag attivi, si riduce a pochi centimetri (20-30cm). Ogni famiglia si suddivide poi in tag di sola lettura e tag scrivibili, il che aumenta notevolmente la loro flessibilità rispetto ai codici a barre. Il costo dei tag passivi di sola lettura si aggira intorno a € 0,70 cadauno se ne compriamo almeno 1000. Tale costo varia comunque in base al quantitativo ordinato ed è comunque destinato a scendere nel tempo cambiando probabilmente ordine di grandezza. Rispetto al codice a barre, la lettura dei tag è molto più comoda data l'assenza di contatto fisico tra il tag ed il lettore. Leggere il codice anche senza che il lettore "veda" l'etichetta, rende questo sistema più veloce e pratico rispetto ai sistemi attualmente utilizzati.

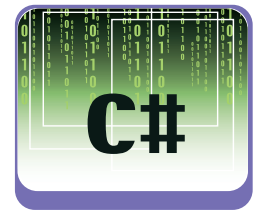
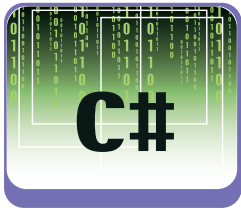
LETTORI RFID

Nel precedente paragrafo abbiamo visto che in commercio esistono diversi tipi di tag, utilizzabili in campi applicativi diversi. I lettori dovranno quindi essere scelti in base alla tipologia di tag che abbiamo intenzione di utilizzare. Il lettore più comune è molto simile ad un lettore di codice a barre tradizionale. In commercio si trovano, oltre ai lettori palmari, anche lettori da tavolo, da parete, da varco ecc., tutti collegabili sia direttamente ad un elaboratore, che a mezzo di opportuni concentratori (un po' come gli HUB in una rete lan). Ciò che differenzia i lettori di tag sono sostanzialmente due fattori: la frequenza di lavoro e la distanza di lettura. I tag, infatti, sono prodotti con moduli di trasmissione a radiofrequenza diversi al fine di rispettare le normative in vigore nei paesi di utilizzo. Sia il lettore che il tag devono dunque "parlare" sulla stessa frequenza. La distanza di lettura è l'altro fattore caratteristico dei lettori di tag. Si va da quelli di prossimità, la cui distanza massima di lettura è di 20/30 cm, a quelli a lungo raggio che, abbinati ai tag attivi (quelli con batteria integrata), permettono la lettura fino a 15/20 metri di distanza. I costi dei lettori variano in base alla tipologia scelta e vanno da un minimo di € 250,00 a salire. Come per i tag, anche il prezzo dei lettori è destinato a scendere proporzionalmente alla loro diffusione.

Fig. 3: ***Un classico lettore di tag palmare.***

In questi articoli vedremo come utilizzare in accoppiata i lettori palmari e i tag passivi.

LEGGIAMO I TAG

Ora che sappiamo cosa sono i tag a radiofrequenza, come funzionano e come si leggono, iniziamo a fare qualche esperimento. In commercio si trovano ormai decine kit per testare questa tecnologia il cui costo si aggira intorno a € 300,00. Mi rendo conto che spendere tale somma per la sola sperimentazione possa scoraggiare qualcu-

Un sito ricco di prodotti relativi a questa tecnologia è http://buyrfid.com/catalog/default.php. Sfogliandolo potrete farvi un'idea dei costi di implementazione nonché delle soluzioni disponibili sul mercato. Considerate sempre che, specie per i tag, i prezzi variano in relazione al quantitativo acquistato.

no quindi, per ovviare al problema, nel codice allegato alla rivista è presente un tool che simula via software la lettura dei tag. Il primo ostacolo che dovremo affrontare è quello relativo alla gestione della porta seriale. Il .net framework non ha, al momento, classi interne *(managed)* per la gestione delle porte seriali. Fortunatamente però attraverso il *Platform Invocation* (*PInvoke* del namespace *System.Runtime.InteropServices*), è possibile utilizzare delle APIs Win32 *(unmanaged)* anche in progetti .net. Basta una ricerca su internet per vedere che molti sviluppatori hanno già realizzato delle librerie pronte all'uso. Essendo un argomento abbastanza vasto, la cui trattazione andrebbe al di fuori dello scopo di questo articolo, ci limiteremo ad usare una di queste librerie commentandola solo quando serve. Tra le tante disponibili in Rete, ho scelto quella scritta da un programmatore Italiano: *Corrado Cavalli, MVP (Most Valuable Professional)* in .net. La libreria di Corrado è scritta in VB.net ma vedremo come sia possibile farlo convivere senza problemi con C#. Creiamo una nuova soluzione in Visual Studio .net relativa al nostro progetto che chiameremo *TagGest.* Al suo interno, realizziamo una nuova libreria di classi Visual Basic che chiameremo *rs232* in cui copieremo il file *rs232.vb* contenuto nel codice allegato.

Il passo successivo sarà quello di creare una piccola applicazione per verificare il corretto funzionamento del sistema. La base di tale applicazione costituirà poi il modulo di ricezione del nostro software. Creiamo dunque una nuova applicazione Windows che chiameremo *TagGest* (come la soluzione), scegliendo questa volta C# come linguaggio. Il form principale deve avere una texbox in cui leggeremo i dati in arrivo sulla seriale, ed un bottone che ci permetta di aprire tale porta al fine di riceverne i dati (vedi Fig.5).

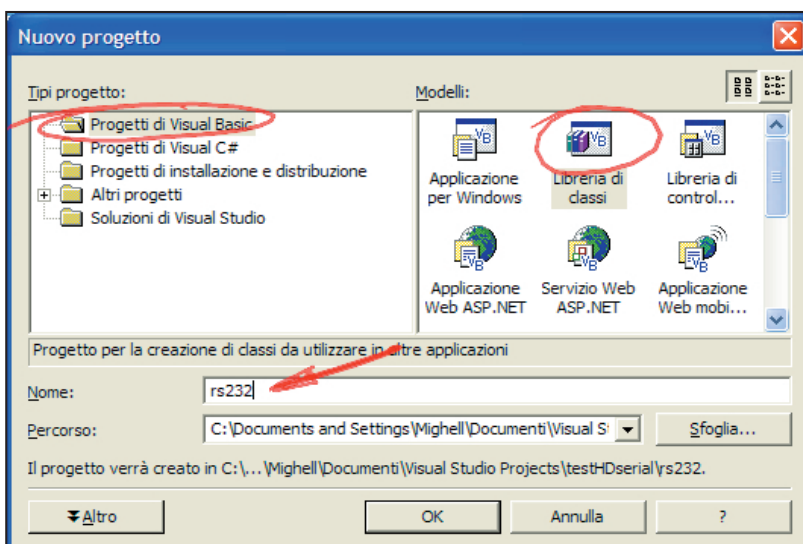



Fig. 4: *Creazione di una Libreria di Classi Visual Basic in Visual Studio .NET.*

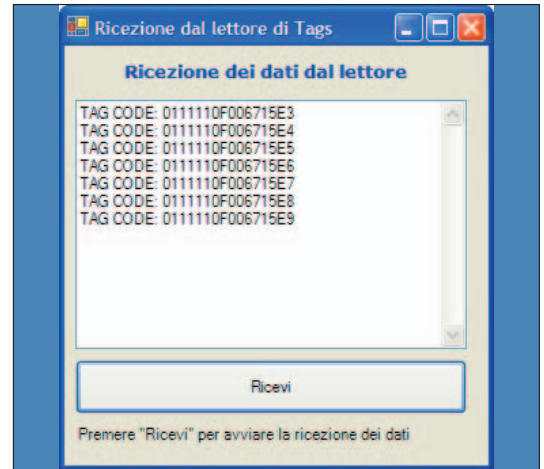



Fig. 5: *Il Form principale del progetto TagGest.*

Per poter usare la libreria di gestione della *rs232* che abbiamo creato all'inizio, dobbiamo innanzitutto referenziarla al progetto *TagGest.* Per farlo, clicchiamo con il tasto destro del mouse sulla cartella *References* del progetto *TagGest*, selezioniamo la voce *"Aggiungi Riferimento"*, scegliamo la cartella *"Progetti"* e selezioniamo *rs232.* Confermata l'operazione, saremo pronti ad usare nel nostro progetto la DLL scritta in VB.net. In *form1.cs*, importiamo il namespace relativo alla *rs232* con la direttiva

```
using rs232;
```

e creiamo una istanza statica di *rs232*

```
private static Rs232 r = new Rs232();
```

L'apertura della porta avviene alla pressione del bottone *"Ricevi"*, avendo però prima settato alcuni parametri fondamentali come la porta su cui ricevere i dati, la velocità ecc.:

```
private void btnRicevi_Click(object sender,
                              System.EventArgs e) {
  try{
    r.Port = 1;
    r.BaudRate = 9600;
    r.DataBit = 8;
    r.StopBit = Rs232.DataStopBit.StopBit_1;
    r.Parity = Rs232.DataParity.Parity_None;
    r.Timeout = 1500;
    r.Open();
    r.Dtr = true;
    r.Rts = true;
    r.RxBufferThreshold = 16;
    r.EnableEvents();
    r.CommEvent+=new
        rs232.Rs232.CommEventHandler(r_CommEvent);
  }catch (Exception ex){
    textBox1.Text = ("Errore: " + ex.Message);} }
```

Questi parametri sono variabili e dipendono dalle specifiche del lettore che verrà utilizzato. Niente paura, in questo esempio i dati sono compilati nel codice ma, nel successivo articolo, vedremo come recuperarli da un file di configurazione in modo da non dover ricompilare l'applicativo in caso di modifica del lettore. Di importanza rilevante è l'abilitazione alla gestione degli eventi attraverso l'istruzione

```
r.EnableEvents();
```

Non dimentichiamoci che la nostra applicazione deve restare in ascolto per ricevere i dati che arrivano dal lettore. Se leggessimo solamente i dati dalla porta seriale con il metodo *read(n)*, leggeremmo solo gli *n* byte che in quel preciso momento stanno arrivando alla porta seriale. Nella migliore delle ipotesi, non riusciremmo a leggere nulla, generando una eccezione di *timeout* dopo un intervallo di tempo pari a quello settato con l'istruzione

```
r.Timeout = 1500;
```

che nel nostro caso specifico è di 1500ms.
Lo scopo dell'istruzione *r.EnableEvents()* è quello di avviare un nuovo *thread* in ascolto sulla porta seriale specificata nei settaggi. Lo vediamo direttamente nella classe *CRs232.vb:*

```
If moEvents Is Nothing Then
   mbEnableEvents = True
   moEvents = New Thread(AddressOf pEventsWatcher)
   moEvents.Start()
End If
```

Quando un numero predefinito di byte (definibile con l'istruzione *RxBufferThreshold*) arriva sulla porta seriale, per il nostro applicativo vorrà dire che abbiamo letto il codice di un tag che quindi dobbiamo processare. Nel caso specifico dell'esempio, il processing che effettueremo sarà quello di visualizzare nella textbox, i dati ricevuti sulla porta. L'assembly *rs232* farà scattare un evento di tipo *CommEvent* a cui avremo registrato il nostro form:

```
r.CommEvent+=new rs232.Rs232.CommEventHandler(
                                    r_CommEvent);
```

che si occuperà, attraverso il delegate *CommEventUpdate*, di aggiornare la *textBox* con il testo:

```
textBox1.AppendText("TAG CODE: " +
                  source.InputStreamString + "\r\n");
```

che rappresenterà il codice appena ricevuto.

Per effettuare i test, colleghiamo il nostro lettore alla porta seriale, apriamo *TagGest.exe*, attiviamo la ricezione e leggiamo i tag con il lettore. In alternativa, colleghiamo il cavo seriale *laplink* a due porte seriali, avviamo il programma di ricezione ed il simulatore (*TagSimulator.exe* contenuto nel codice allegato e comprensivo di sorgenti) ed avviamo la simulazione. Vedremo apparire nella textBox di *TagGest*, i codici inviati dal *TagSimulator*.

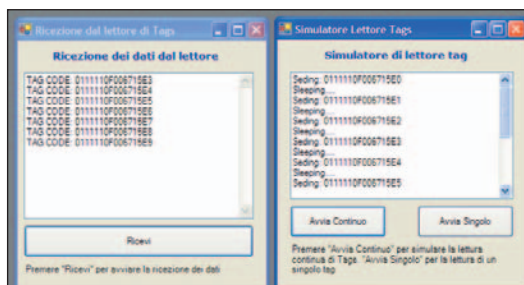
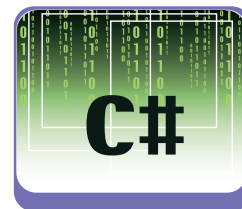



***Fig. 6:** L'applicazione funzionante affiancata al simulatore di lettore di tag*

COSA REALIZZEREMO

In questo primo articolo abbiamo visto cosa la tecnologia ci mette a disposizione per risolvere un problema abbastanza oneroso come la gestione di un magazzino. I tag a radiofrequenza, se usati con criterio, possono rendere più semplice e sicuramente più economica la gestione di un intero magazzino. Abbiamo inoltre fatto la prima sperimentazione di lettura di un tag (reale o simulata), superando quello che è lo scoglio maggiore: l'interfacciamento con un dispositivo Hardware attraverso la porta seriale del pc. Nel prossimo numero vedremo come utilizzare i dati ricevuti per popolare un archivio del nostro magazzino, individuare un prodotto sugli scaffali e fare un vero e proprio inventario in breve tempo. Un ringraziamento particolare va a Corrado Cavalli per la sua disponibilità.

Michele Locuratolo

APPROFONDIMENTI

Per approfondire tutti i dettagli tecnici relativi alla tecnologia dei dispositivi a radiofrequenza utilizzati in questo articolo, consultate il sito http://www.ti.com/tiris/docs/docntr.htm ricco di documentazione, data sheets, white papers.

Se siete interessati all'acquisto di un kit per testare direttamente la tecnologia, ne trovate diversi sul sito http://www.secureorderprocess.com/ti/products.asp. Il kit LF Micro Eval Kit è più che sufficiente per effettuare le sperimentazioni necessarie.

Il progetto completo della libreria di Corrado Cavalli è disponibile sul suo sito: http://www.codeworks.it/net/VBNetRs232.htm. Il progetto è scaricabile liberamente ed è comprensivo di tutti i sorgenti, nonché di una applicazione di test per verificarne l'effettivo funzionamento.

Costruiamo da zero un DBMS

SQL, Java e Design Pattern

La realizzazione di un database in Java è interessante per disporre di un motore da integrare nelle proprie applicazioni, ma anche per apprendere tecniche per la trattazione del testo.

Il database è un componente fondamentale in molte applicazioni e tra tutte le tipologie disponibili è sicuramente più diffuso quello relazionale, che utilizza il linguaggio SQL per creare, cancellare e modificare i dati. Oggigiorno, anche le applicazioni apparentemente più semplici necessitano di un qualche meccanismo per memorizzare le informazioni ed i semplici file binari e di testo possono non bastare più per supportare le necessità dell'applicazione. D'altronde, pare che la nuova versione di Windows non utilizzerà più un normale file system, ma una versione personalizzata di SQL Server.

IL PROGETTO JAVADB

In questa breve serie di due articoli verrà affrontata la realizzazione di un semplice motore di database SQL sviluppato completamente in Java e pensato per essere inserito all'interno di applicazioni standalone. Spesso i database server vivono come processi separati, preferibilmente in esecuzione su macchine dedicate; il progetto javadb qui presentato vive invece all'interno del processo principale dell'applicazione. La realizzazione di un database relazionale non è affatto semplice e ci sono già progetti open source che hanno obbiettivi simili, ma sono più avanti nello sviluppo lo scopo qui è piuttosto quello di affrontare problemi non banali e le relative soluzioni. I due elementi principali su cui si basa javadb sono il parser dei comandi, basato sulla classe *StringTokenizer* e la struttura a comandi, sviluppata secondo il pattern Command.

ELABORARE I COMANDI SQL

Il parser implementato è abbastanza semplice e si basa sulla classe *ParserTokenizer* che non è altro che una sottoclasse di *StringTokenizer*; il fatto di avere una classe tra *StringTokenizer* e le classi client consente di avere un punto dove eventualmente inserire del codice personalizzato per elaborare il linguaggio SQL. I comandi supportati in questa versione sono:

```
DROP DATABASE;
CREATE DATABASE;
CREATE TABLE;
INSERT INTO
SELECT;
```

Ciascuno di questi comandi, come noto ha una diversa sintassi, che viene approfondita in classi comando specifiche; per capire quale di queste invocare viene analizzato l'inizio del comando *(DROP, CREATE, INSERT):* sostanzialmente il primo token della stringa. Se infatti si ha l'istruzione SQL:

```
CREATE DATABASE Prova
```

i token risultanti dall'elaborazione con *StringTokenizer* sono:

```
CREATE
DATABASE
Prova
```

Il parser trova dunque i dati già scomposti e pronti per l'elaborazione, che avviene nella classe *CommandParser*:

```
public Object execute( String sql ) throws SQLException {
  ParserTokenizer st = new ParserTokenizer( sql );
  if( st.hasMoreTokens() ) {
    String commandName = (String)st.nextToken();
    commandName = commandName.toUpperCase();
```

```
    CommandsUtil.trace( commandName );
    Class clazz = (Class)commands.get( commandName );
    if( clazz == null ) {
        throw new SQLException(
            "Comando non supportato: " + commandName); }
    try {
        Command command = createCommand( clazz );
        command.execute( session, st );
        return command.getResult();
    } catch( Exception ex ) {
        ex.printStackTrace();
        throw new SQLException("Impossibile eseguire il
                                            comando");}
  } return null;
}
```

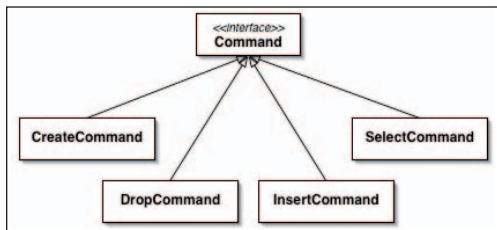



Fig. 1: ***Struttura a comandi***

Come si nota dal codice, in *commandName* viene memorizzato il primo token del comando e viene cercato all'interno della mappa *commands*; se viene trovato, viene eseguita una istanziazione dinamica della classe collegata, su cui viene chiamato il metodo *execute()*. Il risultato viene ottenuto invece con la chiamata al metodo *getResult()*. L'inizializzazione della mappa avviene attraverso il metodo *init()*, che per ciascun comando specifica la classe che ne effettua la gestione:

```
void init() {
  commands = new HashMap();
  commands.put("DROP", it.bigatti.javadb.commands.
                              DropCommand.class);
  commands.put("CREATE", it.bigatti.javadb.commands.
                              CreateCommand.class);
  commands.put("INSERT", it.bigatti.javadb.commands
                              .InsertCommand.class);
  commands.put("SELECT", it.bigatti.javadb.commands
                              .SelectCommand.class);
  //commands.put("DELETE", it.bigatti.javadb.commands
                              .DeleteCommand.class); }
```

Ciascun comando supportato dal programma è implementato da una classe specifica, che implementa l'interfaccia Command (Fig. 1):

```
public interface Command {
  void execute(Session session, ParserTokenizer st)
                                throws SQLException;
  Object getResult(); }
```

CANCELLAZIONE DATABASE: DROP

L'istruzione *DROP DATABASE* è implementata nella classe DropCommand; il metodo execute() verifica il secondo token del comando, cercando DATABASE o TABLE (attualmente non implementato):

```
public class DropCommand implements Command {
  public void execute(Session session, ParserTokenizer st)
    throws SQLException {
      if( st.hasMoreTokens() ) {
        String what = (String)st.nextToken();
        CommandsUtil.trace( "::"+what );
      if( "DATABASE".equals( what ) ) {
          dropDatabase( session, st.nextToken() );
      } else if( "TABLE".equals( what ) ) {}
  } }
}
```

Ovviamente sarebbe opportuno, in una realizzazione più completa, sollevare una eccezione nel caso l'istruzione non sia della sintassi corretta. La cancellazione del database viene effettuata dal metodo *dropDatabase()*, che implementa i seguenti passaggi:

- determinazione del percorso del file dati;
- cancellazione delle tabelle presenti nel database;
- cancellazione del database.

```
void dropDatabase( Session session, String name )
                                  throws SQLException {
  name = CommandsUtil.normalizeName( name );
  CommandsUtil.trace( ":: db="+name );
  File databaseFile = new File(CommandsUtil.getDataPath()
                                  + File.separator +name);
  if( !databaseFile.exists() ) {
    throw new SQLException("Il database " + name + "
                                      non esiste" );}
  String[] files = databaseFile.list();
  for( int i=0; i<files.length; i++ ) {
    File file = new File( databaseFile, files[i] );
    if( !file.delete() ) {
      throw new SQLException("Impossibile eliminare il
        database " + name +" ( impossibile eliminare "
                              + files[i] + " )" );} }
  if( !databaseFile.delete() ) {
    throw new SQLException("Impossibile eliminare il
                              database " + name  );}
}
```

I database e le tabelle vengono memorizzate in modo molto semplice sul file system, utilizzando una directory per ciascun database e due file per ciascuna tabella. Ad esempio:

```
~/.javadb
  + Prova
    Prodotti.data
    Prodott.struct
  + System
    Users.data
```

```
Users.struct
Grants.data
Grants.struct
```

La struttura delle tabelle viene memorizzata come oggetto serializzato, nel file con estensione *.struct*. I dati della tabella sono memorizzati invece nel file con estensione *.data*.

CREAZIONE DEL DATABASE

L'istruzione di creazione delle tabelle e database è invece implementata nella classe *CreateCommand*. Il metodo *createDatabase()*, seguendo la logica di memorizzazione dei dati sopra esposta, non fa altro che creare la directory che ospiterà il database, controllando eventuali errori in fase di creazione, e che il database richiesto non esista già:

```
void createDatabase( Session session, String name )
                                   throws SQLException {
  name = CommandsUtil.normalizeName( name );
  CommandsUtil.trace( ":: db="+name );
  File databaseFile = new File(CommandsUtil.getDataPath()
                                  + File.separator +name);
  if( databaseFile.exists() ) {
    throw new SQLException("Il database " + name + "
                                        esiste già" );}
  if( !databaseFile.mkdirs() ) {
    throw new SQLException("Impossibile creare il
                                  database " + name );}
    session.setCurrentDatabase(new DatabaseData(name));
}
```

Anche in questo caso eventuali errori sono segnalati come *SQLException*; in realtà, essendo ancora sul lato dell'implementazione interna del database e non a livello di interfacce JDBC, sarebbe stato opportuno utilizzare un'eccezione personalizzata, come ad esempio *DatabaseException*, da convertire poi in una *SQLException*. In questo modo si sarebbe realizzata una maggiore divisione tra gli strati dell'applicazione.

CREAZIONE DI UNA TABELLA

Le difficoltà che si incontrano nella creazione di una nuova tabella sono legate alla decodifica dell'istruzione *CREATE TABLE*. Ad esempio:

```
CREATE TABLE Prodotti (id int(11) NOT NULL,
                nome varchar(100), valore number(13,5))
```

Ciascun campo deve essere analizzato per ottenerne il nome, il tipo e la dimensione. Queste informazioni verranno poi memorizzate nella struttura della tabella e potranno essere utilizzate per pilotare le successive operazioni. Come accennato, la struttura del database viene memorizzata in un file serializzato; il nome del file viene ottenuto dalle classi di modello, che contengono i metadati a proposito dello schema del database. Dalla sessione corrente si ottiene il database a cui si è attualmente connessi:

```
DatabaseData databaseData = session.getCurrentDatabase();
```

poi è necessario creare un oggetto che rappresenti la tabella:

```
TableData tableData = new TableData( name, fields, order);
```

a questo punto sono disponibili tutte le informazioni per costruire il percorso del file da creare (p.e. *~/.javadb/Prova/Prodotti.struct*):

```
File tableFile = new File(CommandsUtil.getDataPath() +
        File.separator +databaseData.getFileName() +
            File.separator +tableData.getFileName());
```

l'oggetto *TableData* viene poi serializzato e scritto su disco:

```
try {
  ObjectOutputStream out = new ObjectOutputStream(
                     new FileOutputStream(tableFile));
  out.writeObject( tableData );
  out.close();
} catch( IOException ex ) {
  throw new SQLException("Impossibile creare la tabella "+ex); }
```

Al costruttore di *TableData* è stata passata la variabile fields, che contiene una mappa con i campi del database; per estrarre queste informazioni viene implementato un ciclo, che continua l'iterazione all'interno del comando SQL, utilizzando però una stringa di delimitatori personalizzata, che oltre agli spazi vuoti include le parentesi, in modo che token come *int(11)* vengano ritornati come due elementi diversi: int e 11. Per sapere in ogni momento quale elemento è in fase di lettura, viene utilizzata la variabile fieldItem che contiene un valore numerico diverso per ogni elemento. Ad esempio:

token =	id	int	11	NOT	NULL
item =	1	2	3	4	5

Si noti che la clausola *NOT NULL* è opzionale e potrebbe non essere presente; il codice tiene in considerazione questo fatto ed eventualmente imposta fieldItem a *FD_NONE*, che è una costante che rappresenta lo stato iniziale (si noti che per ciascun elemento da leggere è presente una

NOTA

SQL

SQL è il linguaggio standard per l'accesso ai database relazionali, supportato dai principali database server; sebbene esista uno standard (il più diffuso è ANSI-92) ogni produttore implementa una propria versione del linguaggio derivata da quella standard.

HSQLDB

Il database HSQL (http://hsqldb.sourceforge.net/) è un relazionale scritto in Java, dotato di driver JDBC che supporta un ricco subset delle specifiche ANSI-92; il JAR è piccolo (circa 160KB) e può memorizzare i dati sia su file che in memoria. HSQL è integrabile nelle applicazioni, oppure può essere eseguito come processo server; inoltre include un web server minimale e strumenti di amministrazione. Il codice viene dichiarato come di qualità pronta per la produzione ed è utilizzato per la persistenza in diversi prodotti open source e commerciali.

costante che inizia per *FD_*).

```
while( st.hasMoreTokens() ) {
  String token = (String)st.nextToken(" \t\n\r\f()");
  //toglie eventuali virgole rimaste appese al token
  if( token.endsWith(",") ) {
    token = token.substring(0,token.length()-1);}
  switch( fieldItem ) {
    /**
    ...altro codice
    */
    case FD_NONE:
      fields.put( currentField.name, currentField );
      order.add( currentField );
      fieldItem++;
      break;
    case FD_NAME:
      currentField = new FieldData();
      currentField.name = token;
      fieldItem++;
      break;
    case FD_TYPE:
      currentField.type = token;
      fieldItem = FD_SIZE;
      break;
    case FD_SIZE:
      currentField.size = token;
      fieldItem = FD_NOT;
      break; } }
 if( currentField != null ) {
  fields.put( currentField.name, currentField );
  order.add( currentField ); }
}
```

MODELLO INFORMATIVO

In Fig. 2 è presente il modello che contiene le meta-informazioni sul database, e che è composto dalle classi *DatabaseData*, *TableData* e *FieldData*; queste mantengono, rispettivamente, le informazioni su uno specifico database, su una specifica tabella e su uno specifico campo. Per semplicità queste classi utilizzano attributi pubblici, ma in una versione più evoluta dovrebbero aderire alle specifiche Javabeans ed utilizzare getter e setter. L'oggetto *DatabaseData* mantiene il nome del database, ed offre un metodo per ottenere l'oggetto *TableData* relativo ad una tabella di cui è noto il nome. Questi oggetti vengono inoltre mantenuti in memoria, in una mappa:

Fig. 2: Metadati di javadb

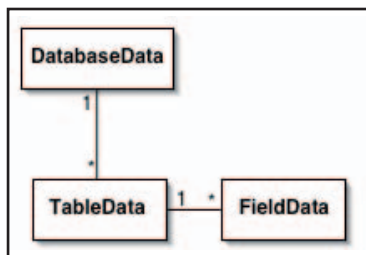
```
public class DatabaseData implements Serializable {
 public String name;
 Map tables = new HashMap();
 public String getFileName() {
   return name; }
 public String toString() {
   return "DatabaseData=[name="+name+
                              ", tables="+tables+"]";}
 public TableData getTable( String name ) throws
                                        SQLException {
 TableData tableData = (TableData)tables.get(name);
   if( tableData == null ) {
     tableData = loadTableData( name );
     tables.put( name, tableData );}
   return tableData; }
 public TableData loadTableData( String name )
                               throws SQLException {
   TableData tableData = null;
   File tableFile = new File(CommandsUtil.getDataPath()
      +File.separator+getFileName()+File.separator+
                        TableData.getFileName(name));
   try {
     ObjectInputStream out = new ObjectInputStream(
                   new FileInputStream( tableFile ));
     tableData = (TableData)out.readObject();
     out.close();
   } catch( ClassNotFoundException ex1 ) {
     throw new SQLException("Impossibile caricare
                                  la tabella " + ex1 );
   } catch( IOException ex2 ) {
     throw new SQLException("Impossibile caricare
                                la tabella " + ex2 ); }
   return tableData;}}
```

Come si nota, la classe è serializzabile (implementa *Serializable*), in quanto è necessario poterla scrivere su disco; come gli altri oggetti del modello informativo, implementa il metodo toString(), in modo da rendere più agevole lo sviluppo e la ricerca degli errori: stampando l'oggetto, ad esempio con un *System.out.println()*, invece di un criptico *hashCode* si troverà una descrizione completa del contenuto dell'oggetto.

TABELLE E CAMPI

La classe *TableData* contiene, oltre al nome della tabella, l'elenco dei campi della stessa; inoltre, viene memorizzata una lista che elenca l'ordine naturale dei campi come sono indicati in fase di creazione. Il problema è che la mappa che contiene i campi non mantiene l'ordine di inserimento nella stessa: quando si richiede l'enumerazione delle chiavi, vengono ritornate in un ordine diverso da quello di inserimento. Se viene creata la tabella in questo modo:

```
CREATE TABLE Prodotti (id int(11) NOT NULL,
           nome varchar(100), valore number(13,5))
```

e poi ad esempio si esegue un comando come:

```
SELECT * FROM Prodotti
```

I dati ritornati dovranno essere del tipo:

```
+----+-------------+--------+
| id | nome        | valore |
+----+-------------+--------+
|  1 | prodotto1   |     12 |
|  2 | prodotto2   |     24 |
+----+-------------+--------+
```

La classe *TableData* viene implementata in questo modo:

```
public class TableData implements Serializable {
  //... altro codice
  public Iterator getFieldsInNaturalOrder() {
    return order.iterator();}
  public List getFieldNames() {
    computeFieldNames();
    return fieldNames;}
  void computeFieldNames() {
    if( fieldNames == null ) {
      fieldNames = new ArrayList( order.size() );
      Iterator iter = order.iterator();
      while( iter.hasNext() ) {
        FieldData fieldData = (FieldData)iter.next();
        fieldNames.add( fieldData.name );}}}
  public String toString() {
    return "TableData=[name="+name+
                            ", fields="+fields+"]";}}
```

La classe *FieldData* include, oltre alle informazioni relative al nome, tipo e dimensione un metodo che ritorna la dimensione totale del campo. Si noti infatti che la dimensione in SQL può includere la precisione. Ad esempio:

```
INT(10,5)
```

In questo caso la dimensione totale del campo è 10 (5 è il numero dei decimali). Il metodo *getSize()* ritorna la dimensione totale del campo, cioè il numero di byte che verranno dedicati alla memorizzazione della colonna:

```
public class FieldData implements Serializable { //...
  public int getSize() {
    int sizeValue = 0;
    //Prende solo la parte intera (dimensione totale
                                          del campo)
    String tempSize = size;
    int pos = tempSize.indexOf(",");
    if( pos >= 0 ) {
      tempSize = tempSize.substring(0,pos);}
    try {
      sizeValue = Integer.parseInt( tempSize );
    } catch( NumberFormatException ex ) {}
    return sizeValue;} }
```

CONNESSIONI E PROVA

Il punto di ingresso per operare con il database è l'oggetto *Session*, che rappresenta una sessione di operatività con la base dati; una nuova sessione è ottenibile da parte delle classi client tramite il metodo *Session.getInstance()*, che non fa altro che ritornare un nuovo oggetto; ovviamente, una versione più evoluta dovrebbe implementare un qualche metodo di autenticazione, ad esempio tramite un metodo *login()*. La sessione mantiene il database corrente su cui si sta operando, che è impostato in automatico ad esempio dopo una *CREATE DATABASE*, oppure esplicitamente tramite una istruzione di *CONNECT*.

```
public class Session {
  DatabaseData databaseData;
  long created;
  public Session() {
    created = System.currentTimeMillis();
    setCurrentDatabase( new DatabaseData( "Prova" ) );}
  public void setCurrentDatabase( DatabaseData
                                    databaseData ) {
    this.databaseData = databaseData;}
  public DatabaseData getCurrentDatabase() {
    return databaseData;}
  public Object execute( String sql ) throws
                                     SQLException {
    CommandParser parser =
                   CommandParser.getInstance( this );
    return parser.execute( sql );}
  public static Session getInstance() {
    return new Session();}
}
```

A questo punto è possibile eseguire le prime prove. Ad esempio, creando un nuovo database ed una nuova tabella:

```
Session session = new Session();
session.execute("CREATE DATABASE Prova");
session.execute("CREATE TABLE Prodotti (id int(11)
  NOT NULL, nome varchar(100), valore number(13,5))");
```

CONCLUSIONI

In questa prima puntata abbiamo creato una prima infrastruttura per la creazione del nostro database, implementando le funzionalità di creazione di database e tabelle (anche se in forma minimale). Nel prossimo numero effettueremo inserimenti, selezioni e cancellazioni di dati nelle tabelle.

Massimiliano Bigatti

L'utilizzo di design pattern è un elemento fondamentale nella progettazione di software ad oggetti non banale; nel progetto presentato in questo articolo utilizza il pattern *Command* descritto nel libro

• DESIGN PATTERNS *Gamma, Helm, Johnson, Vlissides* (Addison Wesley) 1995

Un framework Open Source per la costruzione di interfacce utente

Grafica super in C++ con wxWidgets

wxWidgets è un framework per lo sviluppo di software con interfaccia grafica in C++. Il vasto numero di classi e l'elevata portabilità fanno di questo pacchetto open source un prodotto di qualità.

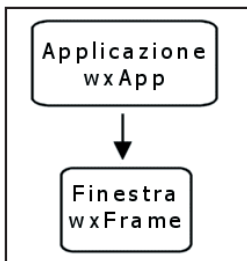



Fig. 1: ***Principali componenti di un'applicazione con interfaccia grafica.***

Scrivere un buon software prevede spesso la realizzazione di un'interfaccia grafica efficiente, l'implementazione di protocolli di rete o il supporto delle funzioni di stampa. Tra i vari framework, che forniscono soluzioni già pronte per questi problemi, wxWidgets spicca per due grandi vantaggi. Oltre ad essere completamente gratuito ed open source vanta un livello di astrazione tale da consentire il porting immediato delle applicazioni verso ognuna delle piattaforme attualmente più diffuse. Esistono infatti versioni di wxWidgets per sistemi Windows, Linux e Mac (per citare i più comuni). Basti pensare che talvolta, per convertire un'applicazione ben scritta, è sufficiente una semplice ricompilazione dei sorgenti con l'opportuna versione del framework.

STRUTTURA DI UN'APPLICAZIONE

Gran parte del codice di wxWidgets è organizzato in classi relative ai vari elementi che servono a mettere insieme l'applicazione. Per esempio, un programma dotato di interfaccia grafica si costruisce con gli elementi che seguono (Fig. 1):

- Un oggetto di applicazione
- Un frame

Per oggetto di applicazione si intende un'istanza della classe *wxApp,* che rappresenta l'applicazione stessa e contiene l'indispensabile per mettere su un programma. Nel codice scritto per *wxWidgets* non appare la procedura main, alla quale siamo abituati, quest'ultima infatti viene sostituita da una funzione membro di nome *OnInit().* Se dunque deriviamo una classe da *wxApp* avremo bisogno di definirla:

```
#include <wx/wx.h>
class myApp : public wxApp
{
public:
    virtual  bool  OnInit();
};
```

OnInit è la funzione del programma che sarà eseguita per prima e che possiamo adoperare per svolgere alcune operazioni preliminari, come la creazione del frame grafico principale.

```
IMPLEMENT_APP(myApp)
bool myApp::OnInit()
{
  wxFrame *MyFrame = new wxFrame(NULL, -1, argv[0]);
  MyFrame->Show(TRUE);
  SetTopWindow(MyFrame);
  return TRUE;
}
```

Il richiamo alla macro *IMPLEMENT_APP* viene inserito per informare wxWidgets dell'esistenza della classe *myApp,* così l'oggetto verrà allocato dinamicamente. Come premesso, OnInit si occupa di creare il frame. Un frame è una finestra che possiamo ridimensionare e posizionare a nostro piacimento.
Il primo elemento passato al costruttore della classe *wxFrame* è un puntatore *NULL* relativo ad un'eventuale finestra *parent,* che per ora non ci interessa. Segue un numero di identificazione, dove -1 indica un parametro di default. All'ultimo argomento sta il nome che sarà visualizzato nella barra del titolo della finestra, qui specificando *argv[0]* facciamo apparire quello che il programma avrà una volta compilato, con il suo path. Si tratta di un membro di wxApp e ripropone perfettamente l'omonimo argomento che di consueto riscontriamo nella procedura main. In pochissime righe abbiamo scritto il nostro primo programma funzionante, congratulazioni! È ancora scarno, ma possiamo abbellirlo con un pezzo di codice da piazzare dopo l'allocazione dell'oggetto *wxFrame*:

```
MyFrame->CreateStatusBar();
MyFrame->SetStatusText("Hello World!");
```

In questo modo aggiungiamo una barra di stato e vi facciamo apparire il classico "Hello World!". Per compilare l'esempio è necessario specificare i file di libreria necessari, a tale proposito vi rimandiamo al materiale allegato all'articolo.

PERSONALIZZIAMO IL FRAME

Dopo avere scritto il primo programmino funzionante andiamo a creare qualcosa di più complesso. Ad esempio, potremmo aggiungere un menu per far accedere l'utente alle funzioni di caricamento e salvataggio file. In wxWidgets, le classi interessate allo scopo sono wxMenu e wxMenuBar. La prima permette di costruire i singoli menu (es: il menu "File" con le voci "Carica", "Salva", "Esci", etc), l'altra invece si occupa di raggrupparli per formare la barra di menu vera e propria, che sarà resa accessibile nella parte superiore della finestra. Per inserire questi due nuovi elementi abbiamo bisogno di derivare una nuova classe da wxFrame:

```
class myFrame : public wxFrame
{ public:
    myFrame(const wxString& titolo);
    ~myFrame();
  private:
    wxMenuBar  *mMenuBar;
    wxMenu      *mFileMenu;
};
```

Derivando da wxFrame abbiamo la possibilità di costruire un frame personalizzato, inserendo le componenti che ci interessano, in questo caso quelle necessarie alla realizzazione del menu. Dopo, la sua creazione può essere effettuata tramite il costruttore della classe myFrame:

```
myFrame::myFrame(const wxString& titolo)
: wxFrame((wxFrame *) NULL, -1, titolo)
{  // Creiamo il menu
   mFileMenu = new wxMenu;
   mFileMenu->Append(FILEOPEN, "&Open File");
   mFileMenu->Append(ABOUT, "&About");
   mFileMenu->Append(EXIT, "&Exit");
   mMenuBar = new wxMenuBar;
   // Lo aggiungiamo alla barra
   mMenuBar->Append(mFileMenu, "&File");
   // Visualizziamo la barra
   SetMenuBar(mMenuBar);
}
```

Per prima cosa invochiamo il costruttore di wxFrame, affinché il frame sia inizializzato regolarmente. Di seguito, allochiamo un oggetto di tipo wxMenu e tramite la funzione membro Append() aggiungiamo le varie voci. FILEOPEN, ABOUT e EXIT sono dei numeri impostati arbitrariamente che, come vedremo, serviranno ad identificare la selezione dell'utente nella fase di gestione degli eventi. Possiamo definire questi valori ricorrendo ad un'enumerazione:

```
enum {
FILEOPEN = 100,
ABOUT = 200,
EXIT = 300
};
```

In fine allochiamo l'oggetto di tipo wxMenuBar ed in un modo molto simile inseriamo il menu creato in precedenza. Richiamando la funzione SetMenuBar() rendiamo visibile la barra di menu.

GESTIONE DEGLI EVENTI

Il menu che abbiamo creato funziona ma non è ancora pienamente operativo, infatti, alla selezione di una determinata voce non segue alcuna azione. Occorre occuparsi della gestione degli eventi. In genere, si parla del verificarsi di un evento quando avviene qualcosa all'interno o all'esterno di un'applicazione. Nel nostro caso, quando l'utente selezionerà una voce dal menu che abbiamo creato, seguirà il presentarsi di un apposito evento che avvertirà dell'accaduto. Tale procedimento in wxWidgets si gestisce con delle tabelle, adoperate per associare una funzione membro della classe a ciascun evento in modo che, quando questo si sarà verificato, sia eseguita la giusta porzione di codice.
Il lettore vedrà subito quanto è semplice nella pratica. Sappiamo che tramite la funzione Append() di wxMenu è possibile associare un numero di identificazione ad ogni voce del menu. Questo numero è lo stesso che impiegheremo per costruire la tabella degli eventi:

```
BEGIN_EVENT_TABLE (myFrame, wxFrame)
  EVT_MENU (FILEOPEN,  myFrame::OnFileOpen)
  EVT_MENU (ABOUT,  myFrame::OnAbout)
  EVT_MENU (EXIT,  myFrame::OnExit)
END_EVENT_TABLE()
```

Il tutto si realizza ricorrendo ad un insieme di macro. BEGIN_EVENT_TABLE() segna l'inizio della tabella; tra i suoi argomenti troviamo il nome della nostra classe myFrame e quello della classe dalla quale è stata derivata, cioè wxFrame. Adoperiamo poi EVT_MENU() per inserire i vari elementi, in questo caso indichiamo a wxWidgets quale funzione mem-

SVILUPPO A COSTO ZERO

È possibile scrivere applicazioni per wxWidgets adoperando come ambiente di sviluppo il noto IDE open source Dev-C++ prodotto da Bloodsheed. Nel CD è presente il materiale necessario per l'integrazione dei due package. Per informazioni http://www.wxwindows.org/devcpp.htm

wxWidgets si adatta a stile e aspetto grafico tipici del sistema per il quale l'applicazione è stata scritta.

Fig. 2: *Un messaggio di about.*

bro della classe va associata al verificarsi di eventi relativi all'uso del menu: è sufficiente specificare il numero di codice dell'evento e il nome della funzione che desideriamo assegnare. La myFrame diventerà poi:

```
class myFrame : public wxFrame
{ // Costrutture
  . . .
  void OnFileOpen(wxCommandEvent& event);
  void OnAbout(wxCommandEvent& event);
  void OnExit(wxCommandEvent& event);
  . . .
  protected:
  DECLARE_EVENT_TABLE()
};
```

Il membro della classe, che tramite *EVT_MENU* associamo di volta in volta all'evento, è una funzione void che ha per argomento un oggetto del tipo *wxCommandEvent*, contenente informazioni sull'evento. Infine, la chiamata alla macro *DECLARE_EVENT_TABLE()* va inserita all'interno di *myFrame*. Ora che sappiamo come associare il codice al verificarsi degli eventi occupiamoci dei casi più semplici. Partiamo con il più elementare, cioè la selezione della voce "Exit" che segna la fine del programma. È sufficiente una chiamata alla funzione Close(): si trova già nella classe, essendo stata ereditata da wxFrame che ne dispone:

```
void myFrame::OnExit(wxCommandEvent& event)
{ Close(TRUE);
}
```

Non appena l'utente selezionerà la voce "Exit" dal menu sarà eseguita in automatico la funzione OnExit ed il programma verrà chiuso.

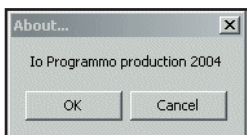
FINESTRE DI DIALOGO

Cogliamo la palla al balzo per aprire un utile parentesi circa le finestre di dialogo. Ad esempio vogliamo fare in modo che, selezionando la voce "About" dal menu, il programma sia momentaneamente sospeso ed appaia una finestra contenente alcune informazioni sul copyright, oppure il nome degli autori (Fig. 2). Per scopi come questo wxWidgets offre un ricco insieme di common dialog. Non andremo a scriverci personalmente i dialog necessari alle varie evenienze poiché i programmatori hanno già pensato di rendere disponibili quelli per le circostanze più comuni, sotto forma di classi pronte all'uso. Per la nostra finestra di about, infatti, è sufficiente adoperare la classe wxMessageDialog, com'è possibile vedere nel codice che segue. Il tutto si risolve in poche righe:

```
void myFrame::OnAbout(wxCommandEvent& event)
{ wxMessageDialog aboutDlg(this,
    "ioProgramma production 2004",
    "About...", wxOK);
  aboutDlg.ShowModal(); }
```

OnAbout è il nome della funzione che, tramite la tabella degli eventi, abbiamo precedentemente destinato allo scopo. Nella prima riga ci limitiamo ad istanziare un oggetto di tipo wxMessageDialog. Qualche riga sopra abbiamo visto che, quando si crea un nuovo frame, è possibile specificare una finestra parent (genitore) al fine di stabilire una gerarchia. Trattandosi allora del frame principale questo dettaglio non ci interessava ed abbiamo messo NULL, indicando così l'assenza di una finestra di livello superiore. Stavolta la situazione è differente poiché possiamo imporre che la finestra parent del nuovo dialog sia proprio il frame principale dell'applicazione. A tale scopo passiamo il puntatore this al primo argomento del costruttore. Seguono il testo del messaggio vero e proprio da visualizzare, il titolo da dare alla finestra che lo conterrà ed il tipo di pulsanti che intendiamo far apparire: wxOK indica un pulsante con su scritto "Ok", un'altra possibile scelta è wxCANCEL. E' consentito visualizzarli entrambi con un'operazione di Or, in C++ si scrive: wxOK | wxCANCEL. Inoltre, ShowModal() visualizza il dialog completo restituendo come valore di ritorno un numero relativo a quella che è stata la selezione dell'utente. Se l'utente ha cliccato il tasto "Ok" la funzione restituisce wxID_OK, altre possibilità sono wxID_CANCEL, wxID_YES: dipende da quali pulsanti si è scelto di visualizzare. Un altro interessante dialog che esaminiamo in questo articolo è quello adottato per la selezione dei file, la classe interessata si chiama wxFileDialog. Immaginiamo di voler far apparire una finestra di tale genere dopo la selezione della voce "Open File" del menu. Scriveremo:

```
void myFrame::OnFileOpen(wxCommandEvent& event)
{  // Allochiamo l'oggetto
  wxFileDialog *openFileDialog = new wxFileDialog
            (this, "Open file", "", "", "*.*" , wxOPEN,
                                        wxDefaultPosition);
  // Visualizziamo il dialog
  if (openFileDialog->ShowModal() == wxID_OK)
  // Prendiamo nome e path del file
  SetFileName(openFileDialog->GetFilename(),
                    openFileDialog->GetDirectory());
}
```

Alla selezione della voce "Open File" del menu segue l'esecuzione della funzione qui proposta. Essa in primo luogo alloca un oggetto di tipo

wxFileDialog deputato alla creazione di una finestra di dialogo per la selezione di files. Nel costruttore vanno specificati l'indirizzo della finestra parent (dunque passiamo this, come visto sopra) ed un messaggio di titolo, seguono due stringhe volte a indicare rispettivamente una directory ed un filename di default (entrambe possono essere vuote, come infatti sono). Dunque, si passa un filtro per i nomi di file (es: *.* oppure *.txt) qui definito tramite una stringa. E' possibile specificare più filtri, ciò è utile se ad esempio vi è la necessita di lavorare con più tipi di file, come nel caso di software per l'elaborazione di immagini che offrono supporto per i vari formati bitmap, jpeg, gif, etc. Questo si attua tramite una sintassi molto semplice: per ogni elemento si specifica una descrizione del tipo di file seguita da un simbolo "|" cui procede il filtro vero e proprio. Per esempio:

```
wxChar *FILTRI = _T("Bitmap|*.bmp|"
                              "JPEG|*.jpg;*.jpeg|""GIF|*.gif");
```

Anche se divisa in più parti per ragioni di comodità, si tratta comunque di un'unica stringa dato che, in C++, le stringhe adiacenti vengono concatenate automaticamente durante la fase di compilazione. Il punto e virgola si adopera quando un tipo di file può presentare differenti estensioni, ciò si verifica in diverse circostanze come nel caso dei file html o dei sorgenti C++ (.cpp, .cc, etc). L'ultimo argomento stabilisce lo stile della finestra, a seconda che si tratti di un dialog di apertura o di salvataggio dei file: si tratta di un valore di tipo long, qui scriviamo wxOPEN. Come per la finestra di about, il dialog viene visualizzato tramite una chiamata alla funzione ShowModal() che restituisce wxID_OK in caso di successo. L'oggetto contiene poi il nome del file selezionato ed il suo path che si ricavano invocando i suoi membri GetFilename() e GetDirectory(). SetFileName è una funzione membro della nostra classe myFrame e serve a memorizzare in un oggetto stringa il nome del file completo del suo path, in modo da ricordarlo se poi lo si dovrà considerare nuovamente, ad esempio per effettuare un salvataggio.

PROGRAMMIAMO UN NOTEPAD

Sappiamo lavorare con le principali classi di wxWidgets, conosciamo come gestire gli eventi ed alcuni common dialog, abbiamo tutte le carte in regola per scrivere il nostro primo programma completo. Per concludere questo articolo di introduzione realizzeremo un semplice editor in stile notepad, con funzioni di caricamento, salvataggio e modifica dei normali file di testo. Qualcuno potrebbe pensare che si tratti di un'operazione complicata gestire un file di testo, caricarlo in memoria, occuparsi della corretta visualizzazione dei caratteri e dei vari elementi di input, ma non è così. Fortunatamente wxWidgets, in modo simile a MFC, offre un insieme di controlli che svolgono in modo semplice e trasparente anche le funzioni che sarebbe più tedioso programmare. Un controllo è, in genere, una piccola finestra che serve a visualizzare un insieme di dati tenendo conto dell'input. Il controllo wxTextCtrl si presta perfettamente al nostro scopo, ci permette addirittura di caricare e salvare file di testo senza saper nulla del sistema di file adoperato da wxWidgets. Tale meraviglia si concretizza inserendo un'unica riga di codice nel costruttore della classe myFrame:

```
mTextCtrl = new wxTextCtrl(this, -1, wxString("Nessun
      file caricato"), wxDefaultPosition, wxDefaultSize,
                                              wxTE_MULTILINE);
```

mTextCtrl è un membro che abbiamo aggiunto a myFrame, si tratta di un puntatore ad un oggetto di tipo wxTextCtrl. Adesso andiamo a parlare del suo costruttore. Innanzitutto abbiamo bisogno che la finestra, dove verrà visualizzato il testo vero e proprio, sia contenuta dentro il nostro frame principale, dunque ancora una volta, specifichiamo il puntatore this; -1 indica il solito parametro di default, la stringa successiva contiene il testo che da principio sarà visualizzato sullo schermo. Attenzione, nessun file da editare è ancora stato caricato. *wxDefaltPosition* e *wxDefaultSize* sono dei parametri che indicano una posizione ed una dimensione standard, infine *wxTE_MULTILINE* fa in modo che la finestra sia volta a contenere un testo disposto su più righe. Non resta che caricare il file mediante la funzione *LoadFile()* di *wxTextCtrl:*

```
mTextCtrl->LoadFile(*mFileName);
```

mFileName è il puntatore alla stringa che contiene il nome del file, che abbiamo inserito tramite la funzione SetFileName(), di cui si è parlato sopra. Il salvataggio si effettua in modo analogo tramite funzione SaveFile(). Ora disponete di un editor di testi funzionante!

CONCLUSIONI

Abbiamo preso in esame alcune delle caratteristiche fondamentali illustrando una via semplice per familiarizzare con questo framework dalle molte possibilità. Invitiamo il lettore ad approfondire l'argomento consultando i sorgenti allegati e visitando il sito di wxWidgets. Buon lavoro!

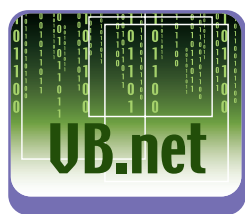
Andrea Ingegneri

SITO WEB
È possibile scaricare versioni più aggiornate di wxWidgets dal sito www.wxwidgets.org

LICENZA
La licenza di wxWidgets 2 permette lo sviluppo di applicazioni sia proprietarie sia open source, le sue condizioni sono valide per scopi personali, accademici o commerciali.

Le procedure per la gestione di un PC in Rete

Controllo remoto in VB (parte ultima)

In questo numero realizzeremo una funzione di chat e daremo la possibilità di condividere risorse. Due funzioni fondamentali per gestire da remoto un PC.

In quest'ultima puntata termineremo il discorso iniziato un paio di mesi fa, analizzando la restante parte del codice implementato. Analogamente alle precedenti puntate, avremo modo di vedere alcune API di Windows molto interessanti e mostreremo alla fine un esempio che ci consente di redirezionare l'output di una finestra DOS verso un controllo di tipo TextBox.

A PROPOSITO DI MOUSE

La volta scorsa abbiamo lasciato il discorso a metà, mostrando semplicemente i primi pulsanti di questa sezione. Vediamo quindi anche i restanti, cominciando da quello che consente di impostare a distanza le coordinate del mouse. La pressione del pulsante *cmdMousePos* non fa altro che provocare l'invio, al server, del messaggio *SETMPOS* che lo processerà, come al solito, attraverso la procedura *ClientRequest()*. Naturalmente, il messaggio *SETMPOS* deve essere seguito da due parametri che rappresentano le coordinate alle quali confinare il cursore del mouse. La tipica stringa inviata al server sarà del tipo:

```
"SETMPOS;" & txtMouseX.Text & ";" & txtMouseY.Text
```

Affinché esso possa "esaudire" tale richiesta, viene invocata un'API di Windows denominata *SetCursorPos()* così dichiarata:

```
Declare Function SetCursorPos Lib "user32" Alias
    "SetCursorPos" (ByVal x As Long, ByVal y As Long)
                                            As Long
```

A questo punto, non è certamente difficile desumere quale sarà il comando invocato dal server, ossia:

```
SetCursorPos ClientRequestSplit(1), ClientRequestSplit(2)
```

Il successivo pulsante di questo pannello permette di effettuare il logoff della sessione corrente. Il messaggio inviato dal client è, come visto nella precedente puntata, *LOGOFF.* Il server, anche questa volta, si serve di un'API di Windows per effettuare quest'operazione, denominata *ExitWindowsEx()*. La sintassi di questa funzione è la seguente:

```
Declare Function ExitWindowsEx Lib "user32" Alias
    "ExitWindowsEx" (ByVal uFlags As Long, ByVal
                        dwReserved As Long) As Long
```

Il primo parametro è quello più importante ed identifica il tipo di *"shutdown"* che si vuole effettuare. Esistono diverse costanti utili allo scopo e sono utilizzabili in varie combinazioni, consentendo di effettuare un restart, uno shutdown, un logoff, ecc. Per i nostri scopi è sufficiente sfruttare la costante *EWX_LOGOFF* (che assume valore 0), impostando il secondo parametro dwReserved al medesimo valore. Il successivo *CommandButton* è quello che si occupa di disabilitare o abilitare la sequenza di tasti *CTRL+ALT+CANC.* A seconda del tipo d'operazione che deve essere compiuta, la parte client del progetto invia un messaggio *ENABLE* o *DISABLE* per indicare al server l'abilitazione o la disabilitazione di questa funzionalità. Dal lato server il tutto si traduce nell'uso di un'altra API, denominata *SystemParametersInfo()*. Questa funzione consente di ottenere informazioni o impostare alcuni parametri relativi all'intero sistema. Il suo uso, per alcuni aspetti, è piuttosto complesso poiché coinvolge moltissime costanti ed impostazioni del sistema. La sintassi di *SystemParametersInfo()* è la seguente:

```
Declare Function SystemParametersInfo Lib "user32"
    Alias "SystemParametersInfoA" (ByVal uAction As Long,
```

```
ByVal uParam As Long, ByVal lpvParam As Any, ByVal
                              fuWinIni As Long) As Long
```

In particolare, il primo parametro identifica l'impostazione sulla quale ottenere informazioni o da reimpostare. I restanti, invece, dipendono dal primo e dall'operazione che si sta compiendo. Poiché sarebbe troppo lungo entrare nel merito, vedremo semplicemente l'istruzione che consente di ottenere quanto vogliamo:

```
SystemParametersInfo(97,Flag,CStr(1),0)
```

dove *Flag* può assumere valore *True* o *False* a seconda se decidiamo di abilitare o meno questa funzionalità.

LA CONDIVISIONE DELLE RISORSE

Siamo finalmente giunti all'ultimo pulsante di questa sezione, il tasto *Share.* La pressione di questo pulsante permette di condividere una risorsa sul computer remoto. Dal momento che l'operazione che si vuole compiere non è molto semplice, per i tanti aspetti che la contraddistinguono, si è preferito creare sul server un modulo a parte, distinto dal *Generale.bas,* ove "confinare" tutte le funzioni, le procedure e le strutture utili allo scopo. Il nome del modulo è Share.bas e contiene molto di più di quanto sia stato realmente utilizzato all'interno del progetto (in particolare, mi riferisco alla parte riguardante la dichiarazione delle strutture e delle costanti utili ad alcune API di Windows).Per cominciare, comunque, ecco la stringa che il client invia al server alla pressione del pulsante *Share:*

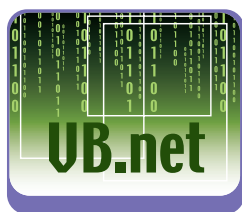
```
"SHARE;" & txtPathShare.Text & ";" & txtNomeShare.Text
```

Il primo parametro indica il percorso della risorsa da condividere, mentre il secondo rappresenta il nome che assumerà la risorsa condivisa. Il server gestisce questo particolare messaggio servendosi della solita procedura *ClientRequest().* Una volta riconosciuto il messaggio, invoca immediatamente la funzione *MakeShare(),* inviando al seguito un codice di risposta al client che lo avvertirà circa l'esito dell'operazione. La funzione *MakeShare()* è così definita:

```
Public Function MakeShare(PathOfShare As String,
                              NameOfShare As String) As Long
  Dim SI502 As SHARE_INFO_502
  Dim SI50 As SHARE_INFO_50
  Dim OSVERInfo As OSVERSIONINFO
  Dim ShareComment As String
  Dim CodErr As Long
  Dim pwd As String
  Dim ret As Long
  'Impostiamo il commento e la password della risorsa
    ShareComment="Remote Share - IoProgrammo"
    pwd="RC1"
  If OSVERInfo.dwPlatformId=1 Then
    'Se Windows 9x...
      SI50.shi50_netname=NameOfShare
      SI50.shi50_path=PathOfShare
      SI50.shi50_remark=ShareComment
      SI50.shi50_type=STYPE_DISKTREE
      SI50.shi50_ro_password=vbNullChar
      SI50.shi50_rw_password=vbNullChar
      CodErr=NetShareAdd9x(0&, 50, SI50, ret)
  Else
    '... altrimenti
      SI502.shi502_current_uses=0
      SI502.shi502_max_uses=10
      SI502.shi502_netname=StrConv(NameOfShare,
                                          vbUnicode)
      SI502.shi502_passwd=StrConv(pwd, vbUnicode)
      SI502.shi502_path=StrConv(PathOfShare,
                                          vbUnicode)
      SI502.shi502_permissions=ACCESS_ALL
      SI502.shi502_remark=StrConv(ShareComment,
                                          vbUnicode)
      SI502.shi502_reserved=0
      SI502.shi502_security_descriptor=Security
      SI502.shi502_type=STYPE_DISKTREE
      CodErr=NetShareAddNT(0&, 2, SI502, ret)
  End If
  MakeShare=CodErr
End Function
```

Gli unici parametri accettati dalla funzione sono *PathOfShare* e *NameOfShare* (che ovviamente rispecchiano proprio i due dati che il client passa al server al momento della richiesta) e restituisce il valore zero in caso d'esito positivo. All'inizio della funzione, vengono dichiarate due variabili "struttura" denominate rispettivamente *SI502* ed *SI50.* Esse rispecchiano le seguenti due strutture:

Fig. 1: La condivisione della directory C:\WinXP.

```
Public Type SHARE_INFO_502
    shi502_netname As String
    shi502_type As Long
    shi502_remark As String
    shi502_permissions As Long
    shi502_max_uses As Long
    shi502_current_uses As Long
    shi502_path As String
    shi502_passwd As String
```

```
        shi502_reserved As Long
        shi502_security_descriptor As
                                    SECURITY_DESCRIPTOR
End Type
```

Questa la seconda struttura:

```
Type SHARE_INFO_50
      shi50_netname As String
      shi50_type As String
      shi50_flags As Long
      shi50_remark As String
      shi50_path As String
      shi50_rw_password As String
      shi50_ro_password As String
End Type
```

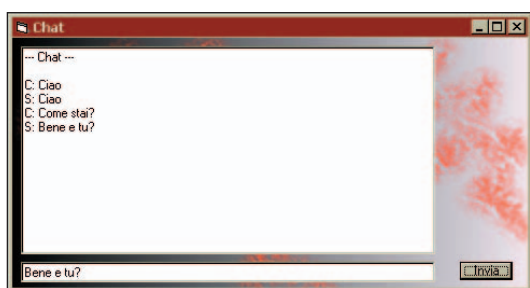



Fig. 2: La finestra Chat (lato server).

Come potrete facilmente intuire, queste due strutture serviranno da "base dati" per alcuni funzioni appartenenti alle API di Windows. In particolare, le due variabili *ShareComment* e *pwd*, valorizzate all'inizio della funzione, consentono d'impostare rispettivamente, un commento e la password d'accesso della risorsa che si sta per condividere. Dettagli a parte, il punto fondamentale del codice relativo a *MakeShare()*, è innanzitutto l'identificazione del sistema operativo avviato sull'host/server poiché sarà questo a determinare l'utilizzo di una delle due strutture precedenti. Per far questo, la funzione *MakeShare()* si serve dell'API *GetVersionEx()* la cui sintassi è:

```
Declare Function GetVersionEx Lib "kernel32" Alias
            "GetVersionExA" (ByVal lpVersionInformation
                                As OSVERSIONINFO) As Long
```

L'unico parametro accettato, *lpVersionInformation*, come si può osservare, è una variabile di tipo *OSVERSIONINFO*, una struttura in grado di ritornare diverse informazioni a riguardo. Essa risulta così definita:

```
Public Type OSVERSIONINFO
        dwOSVersionInfoSize As Long
        dwMajorVersion As Long
        dwMinorVersion As Long
        dwBuildNumber As Long
        dwPlatformId As Long
        szCSDVersion As String * 128
End Type
```

In particolare, *MakeShare()* si preoccupa di controllare l'item *dwPlattformId*. Esso può assumere diversi valori che condizioneranno il comportamento successivo della funzione. Infatti, nel caso il valore ritornato sia pari ad uno, vene riempita opportunamente la struttura *SI50* dichiarata in testa alla funzione, altrimenti viene valorizzata la *SI502*. Esiste anche la possibilità (remota…) che si tratti di sistemi operativi *Windows For Workgroup*. In tal caso, la *GetVersionEx()* ritornerebbe, come valore dell'item *dwPlatformId*, il valore zero. A questo punto, appurato il tipo di sistema operativo, la funzione *MakeShare()* opera alcune scelte. Nel caso *dwPlatformId* risulti essere pari ad uno, verrà sfruttata la *NetShareAdd9x()* così dichiarata:

```
Public Declare Function NetShareAdd9x Lib "svrapi.dll"
        Alias "NetShareAdd" (ByVal servername As Any,
          ByVal slevel As Long, buf As SHARE_INFO_50,
                          ByVal cbbuf As Long) As Long
```

che opererà sulla struttura *SI50* precedentemente valorizzata. Nel caso *dwPlatformId* sia, invece, diverso da uno, sfrutterà l'API *NetShareAddNT()* la cui sintassi risulta essere praticamente la stessa, fatta eccezione per il terzo parametro, *buf*, che non è più una struttura *SHARE_INFO_50*, ma *SHARE_INFO_502:*

```
Public Declare Function NetShareAddNT Lib "netapi32.dll"
      Alias "NetShareAdd" (ByVal servername As Any, ByVal
      slevel As Long, buf As SHARE_INFO_502, ByVal cbbuf
                                       As Long) As Long
```

Il valore di ritorno di entrambe queste funzioni, qualunque sia quella utilizzata, verrà quindi sfruttato da *MakeShare()* per informare il client circa l'esito dell'operazione, permettendogli di mostrare a video il corretto messaggio.

LA SEZIONE CHAT

La sezione *Chat* è probabilmente la sezione più semplice che sia stata implementata all'interno del programma. Attraverso essa, infatti, il client avverte il server circa l'intenzione di avviare una sessione di questo genere ed esso, riconosciuto il messaggio, mostra a video un'apposita finestra utile a tale colloquio. Quando il client decide di avviare una sessione di chat, invia al server un messaggio *AVVIO* attraverso il controllo *WinsockChat*, sulla porta definita da *PORT_RC+1*. Il server, a questo punto, riceve la richiesta attraverso l'evento *DataArrival()* dell'omonimo controllo *WinsockChat*, che è così definito:

```
Private Sub WinsockChat_DataArrival(ByVal bytesTotal
                                          As Long)
   WinsockChat.GetData RispostaChat, vbString
```

```
Select Case RispostaChat
    Case "AVVIO":
        frmChat.Show
        EventLogChat.AddItem "Inizio sessione Chat"
        Exit Sub
    Case "CLOSE":
        Unload frmChat
        EventLogChat.AddItem "Fine sessione Chat"
        Exit Sub
End Select
frmChat.txtChat.Text=frmChat.txtChat.Text+vbCrLf
                                    +"C: "+RispostaChat
End Sub
```

Per prima cosa, mostra a video la form *frmChat,* poi aggiorna successivamente il controllo *EventLogChat* con la stringa *Inizio sessione Chat,* dando inizio allo scambio d'informazioni. In particolare, si osservi che esso riconosce semplicemente due tipi di messaggi ossia *AVVIO* (per iniziare la sessione) e *CLOSE* (per terminarla). Tutto quello che riceve e non equivale a nessuno di questi due prefissi, viene considerato come un messaggio della chat e viene accodato ai precedenti attraverso l'ultima istruzione. Essa, in particolare, antepone ad ogni messaggio, la stringa "C: " che consente di distinguere un messaggio del server da quello del client. Analogamente a quanto accade per il server, un suo messaggio, viene "catturato dall'evento *DataArrival()* del controllo *WinsockChat.* Il codice che implementa le funzionalità di chat, lato client, è molto più semplice e simile a quanto visto precedentemente.

```
Private Sub WinsockChat_DataArrival(ByVal bytesTotal
                                            As Long)
    'Acquisisce la risposta dal server e
    'la memorizza in RispostaChat
        WinsockChat.GetData RispostaChat, vbString
        txtChat.Text=txtChat.Text+vbCrLf+"S: "+RispostaChat
End Sub
```

Anch'esso, ad ogni messaggio ricevuto, antepone una "S: " (che permette di distinguerlo dai messaggi inviati) e l'accoda ai restanti contenuti all'interno della TextBox *txtChat.* Contrariamente a quanto visto nella parte server, non è stato previsto un messaggio di tipo *CLOSE* per consentire al server d'interrompere la sessione di chat. Ovviamente, questa è solo una "personale" convenzione e chiunque può decidere di variarla modificando il codice poc'anzi presentato affinché riconosca e gestisca tale messaggio.

LA SEZIONE EXPLORER

Siamo giunti finalmente all'ultima sezione, denominata *Explorer.* In questa sezione sono state raccolte alcune informazioni importanti circa il sistema server al quale ci si è connessi ed in particolare:

- *Nome del PC;*
- *Directory corrente;*
- *Elenco di alcune variabili d'ambiente (OS, TEMP, USERNAME, CPU Identifier e CPU Level).*

Ovviamente, questa sezione poteva essere molto più complessa e completa allo stesso tempo e poteva includere qualunque tipo d'informazione. Tuttavia, semplicemente per non appesantire troppo il codice presentato, si è preferito mostrare solo qualche esempio, lasciando che gli interessati modifichino il codice a proprio desiderio. Con la pressione del pulsante *cmdSezExplorer* viene inviato al server un messaggio *INFO.* Il server processerà questa richiesta attraverso la "solita" *ClientRequest()* e demanderà il compito di gestirla alla procedura *InviaInformazioni()* contenuta all'interno del modulo *Generale.bas.* Essa risulta così strutturata:

Fig. 3: ***La finestra Chat (lato client).***

```
Public Sub InviaInformazioni()
    Dim BufLen As Long
    Dim StrTemp As String
    Dim DataInfo As String
    'Inizializza DataInfo che, a procedura ultimata
    'verrà inviata al socket client
        DataInfo="INFO;"
'— Nome PC —————————————————————
    'Crea il buffer
        BufLen=MAX_COMPUTERNAME_LENGTH+1
        StrTemp=String(BufLen,"X")
    'Ottieni il nome del PC
        GetComputerName StrTemp, BufLen
    'get only the actual data
        StrTemp=Left(StrTemp,BufLen)
    'Aggiorna DataInfo
        DataInfo=DataInfo+StrTemp+";"
'— Directoy corrente—————————————————
    'Crea il buffer
        StrTemp=String(255,0)
    'retrieve the current directory
        GetCurrentDirectory 255,StrTemp
    'Aggiorna DataInfo
        DataInfo=DataInfo+StrTemp+";"
'— Elenco di alcune variabili di ambiente ———————
        StrTemp=String(255,0)
        StrTemp=Environ$("OS")
```

```
    DataInfo=DataInfo+StrTemp+";"
    StrTemp=Environ$("USERNAME")
    DataInfo=DataInfo+StrTemp+";"
    StrTemp=Environ$("TEMP")
    DataInfo=DataInfo+StrTemp+";"
    StrTemp=Environ$("PROCESSOR_IDENTIFIER")
    DataInfo=DataInfo+StrTemp+";"
    StrTemp=Environ$("PROCESSOR_LEVEL")
    DataInfo=DataInfo+StrTemp
    frmServer.WinsockServer.SendData DataInfo
End Sub
```

Fig. 4: *La sezione Explorer.*

Per ritornare le informazioni al client, ancora una volta, viene sfruttato il sistema d'inviare un'unica stringa in cui ogni "valore" è separato da un ";". Innanzitutto, quindi, viene preparato il prefisso di tale messaggio, rappresentato dalla variabile stringa *DataInfo* ed inizializzato con il valore *"INFO;"*. Successivamente ha inizio la rilevazione della prima informazione ossia il nome del PC. Essa è ottenuta attraverso l'uso dell'API *GetComputerName()* e riposta nella variabile temporanea *StrTemp*. Ad esecuzione ultimata, tale stringa è accodata a *DataInfo* e "terminata" con il solito ";". Il passo seguente è quello di rilevare la directory di lavoro corrente. *InviaInformazioni()* si serve di *GetCurrentDirectory()* per raggiungere tale scopo e, analogamente a quanto visto in precedenza, accoda l'informazione a *DataInfo*. Il processo va avanti anche per i successivi dati, riguardanti le informazioni su alcune variabili d'ambiente. L'ultimo dato, quello relativo alla variabile d'ambiente *PROCESSOR_LEVEL*, è accodato a DataInfo senza che sia terminata con il solito ";" ed inviata al client attraverso il metodo *SendData* del controllo *WinsockServer*. Al ricevimento di un messaggio con prefisso *INFO*, la procedura *ServerRequest()* del modulo client, provoca una serie di semplici azioni, riassunte di seguite:

```
'Nome PC
  Main.lblNomePC.Caption=ServerRequestSplit(1)
'Directory corrente
 Main.lblCurrDir.Caption=ServerRequestSplit(2)
'Variabile ambiente OS
 Main.lblSO.Caption=ServerRequestSplit(3)
'Variabile ambiente USERNAME
 Main.lblUsername.Caption=ServerRequestSplit(4)
'Variabile ambiente TEMP
 Main.lblVarTEMP.Caption=ServerRequestSplit(5)
'Variabile ambiente CPU IDENTIFIER
 Main.lblCPUIdentifier.Caption=ServerRequestSplit(6)
'Variabile ambiente CPU LEVEL
Main.lblCPULevel.Caption=ServerRequestSplit(7)
```

Inutile sottolineare il significato di queste istruzioni poiché credo sia chiaro a tutti. Naturalmente, una sezione di questo tipo, analogamente a quella denominata *Varie,* si presta molto bene ad ulteriori modifiche ed ampliamenti poiché il "raggio d'azione" è davvero molto ampio.

ALCUNI SUGGERIMENTI

Inutile dire che il programma appena mostrato, avendo uno scopo "didattico", non è certamente completo, né tantomeno vuole esserlo. Tuttavia, prima di lasciarvi, mi preme evidenziare una "piccola" routine che vi consentirà di migliorarlo ancor di più. Essa è stata prelevata direttamente da Internet e potete utilizzarla come meglio credete. Prima di mostrarvela, volevo solo evidenziare un piccolo particolare del programma presentato (lato server) che non è stato reso evidente sinora. Se analizzate bene il codice implementato, vi accorgerete della presenza di un controllo *Timer* sulla form principale.
L'evento *Timer()* legato ad esso possiede al suo interno le seguenti istruzioni:

```
Private Sub Timer1_Timer()
   Dim Pt As POINTAPI
   Dim HandleWin As Long
   'Preleva le coordinate correnti del mouse
     GetCursorPos Pt
   'Handle della finestra sotto il mouse
     HandleWin=WindowFromPoint(Pt.X, Pt.Y)
End Sub
```

Attraverso queste semplici linee di codice, potete venire a conoscenza, in qualunque momento, delle coordinate correnti del mouse e, soprattutto, ottenere l'handle della finestra sulla quale si trova attualmente il cursore. Queste informazioni, specialmente la seconda, sono molto interessanti e vi consentiranno d'implementare qualunque tipo d'azione sfruttando la miriade di API che vogliono, necessariamente, tra i loro parametri, un handle. Naturalmente lascio ad ognuno di voi il compito di sfruttare quanto appena detto e sono certo che ne saprete trarre il giusto beneficio. Abbiamo visto che il programma è in grado di avviare applicazioni ed allo stesso tempo possiamo anche inviare file modificando opportunamente alcune parti del codice implementato, con particolare riferimento alla funzione *SendFile()*.

Francesco Lippo

I trucchi del mestiere

Tips & Tricks

Questa rubrica raccoglie trucchi e piccoli pezzi di codice che solitamente non trovano posto nei manuali, frutto dell'esperienza di chi programma. Alcuni di essi sono proposti dalla redazione, altri provengono da una ricerca su internet, altri ancora ci giungono dai lettori. Chi vuole contribuire potrà inviarci i suoi Tips&Tricks preferiti. Una volta selezionati, saranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

COMBOBOX CON ELEMENTI UNIVOCI

Un semplice fa funzionale modulo per non avere nelle ComboBox parole e frasi ripetute.

Tip fornito dal sig. B.Croce

```
Public Function Doppioni(Parola As String, NomeOggetto As
                                        ComboBox) As String
ce = 0
For doppio = 0 To (NomeOggetto.ListCount)
   If Trim(UCase(NomeOggetto.List(doppio))) Like UCase(
                                        Trim(Parola)) Then
      ce = 1
      Exit Function
   End If
Next doppio
NomeOggetto.AddItem NomeOggetto.Text
End Function
'Disegnare su un form un controllo ComboBox e CommandButton
Private Sub Command1_Click()
'Doppioni 'Combo1.text = la parola da controllare','Combo1 l'oggetto
                                        ComboBox dove deve cercare'
Doppioni Combo1.Text, Combo1
End Sub
```

CREARE UNA DIRECTORY AL VOLO

Un utilissimo tip che mostra come creare al volo una cartella di sistema. La funzione *CreateFolder* di tipo booleano richiede un solo parametro di tipo tinga che corrisponde al percoso della cartella da creare. Un esempio può essere *c:\innoland*. Il valore di ritorno è un booleano ed è *True* se la directory viene creata e *False*

IL TIP DEL MESE

DA TESTO A PDF

Di seguito è mostrato come richiamare correttamente la procedura: *ConvertToPDF "Esempio.txt", "Esempio.pdf", "AutoreEsempioTXT", "CreatoreEsempioTXT", "KeyWordTXT", "OggettoTXT", "TitoloTXT", "Courier", 90, 11, 8.* Tutti i parametri esclusi i primi due non sono obbligatori. Gli ultimi tre campi rappresentano la: *Rotazione* (90,180,270 o 360), *Larghezza* e *Altezza* in pollici. Questo comando fa parte della classe *TXTToPDFbas* che, trovate in versione integrale presente nel CD-Rom allegato alla rivista e sul sito web di ioProgrammo (*www.ioprogrammo.it*).
Di seguito è riportata la sola procedura *ConvertToPDF.*

Tip fornito dal sig. A.Bracci

```
Public Sub ConvertToPDF(filename As String, outputfile As String,
       Optional TextAuthor As String, Optional TextCreator As String,
    Optional TextKeywords As String, Optional TextSubject As String,
        Optional TextTitle As String, Optional FontName As String =
      "Courier", Optional FontSize As Integer = 10, Optional Rotation
        As Integer, Optional pwidth As Single = 8.5, Optional pheight
                                                    As Single = 11)
 On Error GoTo er
 If Not FileExists(filename) Then
   MsgBox "File '" & filename & "' does not exist."
  Exit Sub
 ElseIf FileExists(outputfile) Then
  Kill outputfile
 End If
 Initialize FontName, FontSize, Rotation, pwidth, pheight
 author = TextAuthor
 creator = TextCreator
 keywords = TextKeywords
 subject = TextSubject
 Title = TextTitle
 filetxt = filename
 filepdf = outputfile
 Call WriteStart
 Call WriteHead
 Call WritePages
 Call EndPDF
 Exit Sub
er:
  MsgBox Err.Description
End Sub
```

se vi son stati dei problemi. Il try-catch è stato messo apposta per evitare degli errori e consentire il ritorno del parametro.

Tip fornito dal sig. A.Carratta

```
Public Function CreateFolder(ByVal FolderPath As String) As Boolean
    'Script creato da Carratta Andrea aka innovatel
    'Sito Web: www.innoland.it
    'EMail: innoland@innoland.it
      Dim blResult As Boolean = False
      If (FolderPath.Trim <> "") Then
          Try
              Dim Folder As System.IO.DirectoryInfo
              Folder = New System.IO.DirectoryInfo(FolderPath)
              If Not (Folder.Exists) Then
                  Folder.Create()
                  blResult = True
              End If
          Catch
              blResult = False
          End Try
      End If
      Return blResult
End Function
```

UN REPORT GRAFICO PER LE APPLICAZIONI

Alcune applicazioni necessitano, dopo aver elaborato un certo numero e tipo di dati, di mostrare i risultati ottenuti con l'ausilio di controlli ad alto impatto visivo come i grafici. Esistono numerosi controlli di terze parti che offrono la possibilità di risolvere questo problema ma, con le possibilità che il Framework .NET ci offre, è possibile creare questi controlli personalmente. La procedura allegata, sviluppata in Visual Basic .NET consente la creazione di un controllo per la creazione di semplici grafici, con la possibilià di inserire etichette per ogni punto del grafico, scelta dei colori delle linee, zoom in/out, etc...
Il codice VB.NET è presente nel CD-Rom allegato alla rivista e sul web (*www.ioprogrammo.it*).

Tip fornito dal sig. P.Libro

JAVA

"ALBERI" PERSONALIZZATI

Di seguito viene proposto un modo per utilizzare delle immagini personalizzate negli alberi in Java Swing. Bisogna estendere la classe dei nodi *DefaultMutableTreeNode* per aggiungere un parametro per memorizzare l'indirizzo dell'immagine. (Classe *NodoIcon* nell'esempio). Successivamente, è necessario estendere la classe che si occupa di disegnare le immagini '*DefaultTreeCellRenderer*', per visualizzare le nuove immagini (Classe *IconeAlbero* nell'esempio). Infine, basta utilizzare i nuovi nodi ed impostare la nuova classe che si occuperà di disegnare le immagini (*Classe Swing* nell'esempio). Il codice sorgente è presente nel CD-Rom allegato alla rivista e sul web (*www.ioprogrammo.it*).

Tip fornito dal sig. G.Orsini

```
FILE IconeAlbero
import javax.swing.tree.DefaultTreeCellRenderer;
import javax.swing.ImageIcon;
import java.awt.Component;
import javax.swing.JTree;
public class IconeAlbero extends DefaultTreeCellRenderer
{  //Implementazione metodo
public Component getTreeCellRendererComponent(JTree tree,
        Object value, boolean sel, boolean expanded, boolean leaf,int row,
                                                    boolean hasFocus)
  {  //Richiesta a superclasse
    super.getTreeCellRendererComponent(tree, value, sel,
                                  expanded, leaf, row, hasFocus);
    //Recupera i dati dell'icona
    NodoIcon nodo = (NodoIcon)value;
    ImageIcon imgIcona;
    //Carica l'icona richiesta
    imgIcona=new ImageIcon(nodo.getIcon());
    //Verifica l'esistenza dell'icona e la carica
    if(imgIcona!=null)
      setIcon(imgIcona);
      //Restituisce l'oggetto
    return this; }}
FILE Swing
import java.awt.*;
import javax.swing.*;
public class Swing extends JFrame
{  public Swing()
  {  //Swing di esempio
     //Pannello principale
     final JPanel pan=(JPanel)getContentPane();
     //Lista immagini
     String radice="Icone/Radice.gif";
     String nodo1="Icone/Nodo1.gif";
     String nodo2="Icone/Nodo2.gif";
     String foglia1="Icone/Foglia1.gif";
     String foglia2="Icone/Foglia2.gif";
     String foglia3="Icone/Foglia3.gif";
     //Definizione nodi
     NodoIcon A = new NodoIcon("Radice",radice);
     NodoIcon B = new NodoIcon("Nodo A",nodo1);
     NodoIcon C = new NodoIcon("Nodo B",nodo2);
     //Definizione foglie
     C.add(new NodoIcon("Foglia B1",foglia3));
     C.add(new NodoIcon("Foglia B2",foglia3));
     B.add(new NodoIcon("Foglia A1",foglia1));
     B.add(new NodoIcon("Foglia A2",foglia2));
     B.add(C);
     B.add(new NodoIcon("Foglia A3",foglia1));
     B.add(new NodoIcon("Foglia A4",foglia2));
     //Unione nodi a radice
     A.add(B);
     //Preparazione albero
     final JTree albero = new JTree(A);
     //Impostazione disegno albero
```

```
        albero.setCellRenderer(new IconeAlbero());
        //Preparazione finestra
        setSize(200,300);
        setTitle("Albero");
        pan.setLayout(new BorderLayout());
        //Aggiunta albero a finestra
        pan.add(albero,BorderLayout.CENTER);
        //Gestione chiusura finestra
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); }
    //Inizio
    public static void main(String[]arg)
    {   //Definizione e avvio finestra prova
         Swing f=new Swing();
         f.show(); }}
FILE NodoIcon
import javax.swing.tree.DefaultMutableTreeNode;
public class NodoIcon extends DefaultMutableTreeNode
{ private String icona;
   //Costruttore con specifica icona
   public NodoIcon(String testo,String pathIcona)
   {
      super(testo);
      icona=pathIcona;}
    //Restituisce l'icona attuale del nodo
    public String getIcon()
   { return icona; }
   //Imposta l'icona del nodo
   public void setIcon(String pathIcona)
   { icona=pathIcona;}
}
```

COME "ZIPPARE" UNA DIRECTORY

Attraverso il package *java.util.zip*, Java fornisce alcuni semplici meccanismi per comprimere e decomprimere file. Il metodo *zipDir* mostra come comprimere ricorsivamente un albero di directory. Il metodo accetta un oggetto *String* come directory da comprimere e un oggetto *ZipOutputStream* che rappresenta il file compresso. Il metodo *zipDir* non include le directory vuote nell'archivio zip creato.

```
try {
ZipOutputStream zos = new
ZipOutputStream(newFileOutputStream(".\\curDir.zip"));
zipDir(".\\inFolder", zos);
zos.close(); }
catch(Exception e)  {}
public void zipDir(String dir2zip, ZipOutputStream zos)
{try
{ zipDir = new File(dir2zip);
   String[] dirList = zipDir.list();
   byte[] readBuffer = new byte[2156];
   int bytesIn = 0;
   for(int i=0; i<dirList.length; i++)
  { File f = new File(zipDir, dirList[i]);
     if(f.isDirectory())
     { String filePath = f.getPath();
        zipDir(filePath, zos);
         continue;
     }
     FileInputStream fis = new FileInputStream(f);
     ZipEntry anEntry = new ZipEntry(f.getPath());
     zos.putNextEntry(anEntry);
     while((bytesIn = fis.read(readBuffer)) != -1)
     {
        zos.write(readBuffer, 0, bytesIn);
      }
      fis.close();
    }
}
catch(Exception e)
{
}
```

ESEGUIRE APPLICAZIONI ALL'AVVIO DI WINDOWS

Utile funzione per far si che un'applicazione sia eseguita all'avvio di Windows subito dopo l'installazione di un particolare pacchetto.

```
procedure RunOnStartup(sProgTitle, sCmdLine : string;
                                          bRunOnce  : boolean );
 var
   sKey : string;
   reg  : TRegIniFile;
 begin
   if( bRunOnce )then
     sKey := 'Once'
   else
     sKey := '';
   reg := TregIniFile.Create( '' );
   reg.RootKey:= HKEY_LOCAL_MACHINE;
   reg.WriteString(
     'SoftwareMicrosoft'
     + 'WindowsCurrentVersionRun'
     + sKey + #0,
     sProgTitle,
     sCmdLine);
   reg.Free;
 end;
```

INSERIRE UN FILE JPEG IN UN ESEGUIBILE

Includere un'immagine JPEG nell'eseguibile di un'applicazione è molto semplice, basta utilizzare i file di risorse e il compilatore di risorse Borland (a riga di comando). Con un editor di testi (Notepad) create un file di risorse e aggiungete la riga *RCDATA "Pic.jpg"* salvate il file appena creato come *MyPic.RC* e compilatelo con il *Borland Resource Compiler*

```
BRCC32 MyPic.RC
```

Il compilatore creerà il file *MyPic.RES*, a questo punto aggiungete la direttiva per il compilatore nel sorgente del vostro programma subito dopo la direttiva per il form.

```
{$R *.DFM}
{$R MyPic.RES}
```

Ecco il codice da inserire nell'applicazione.

```
procedure LoadJPEGfromEXE;
var
  MyJPG : TJPEGImage;
  ResStream : TResourceStream;
begin
  try
    MyJPG := TJPEGImage.Create;
    ResStream := TResourceStream.CreateFromID(HInstance, 1,
                                                        RT_RCDATA);
    MyJPG.LoadFromStream(ResStream);
    Canvas.Draw(12,12,MyJPG);
  finally
    MyJPG.Free;
    ResStream.Free;
  end;
end;
```

AVVIARE SERVIZI WINDOWS

In alcuni casi è indispensabile arrestare o avviare servizi, anche remoti, per evitare conflitti con qualche applicazione in esecuzione. Il codice seguente mostra come avviare e arrestare servizi in un sistema Windows.

```
uses WinSvc;
// avvia il servizio e restituisce TRUE se l'operazione è andata a buon fine
// sMachine:  nome della macchina, esempio: \SERVER
// empty = macchina locale
// sService  nome del servizio, esempio: Alerter
function ServiceStart(sMachine, sService : string ) : boolean;
 var
   schm,
   schs   : SC_Handle;
   ss     : TServiceStatus;
   psTemp : PChar;
   dwChkP : DWord;
 begin
   ss.dwCurrentState := -1;
   sch := OpenSCManager( PChar(sMachine), Nil,  SC_MANAGER_CONNECT);
  if(schm > 0)then
   begin
     schs := OpenService(schm, PChar(sService), SERVICE_START or
SERVICE_QUERY_STATUS);
     if(schs > 0)then
     begin
       psTemp := Nil;
       if(StartService(schs, 0, psTemp)) then
       begin
         if(QueryServiceStatus(schs, ss)) then
         begin
           while(SERVICE_RUNNING <>ss.dwCurrentState) do
           begin
                   dwChkP := ss.dwCheckPoint;
                   Sleep(ss.dwWaitHint);
                   if(not QueryServiceStatus(schs,ss)) then
                   begin
                   break;
           end;
           if(ss.dwCheckPoint < dwChkP) then
           begin
             break;
           end;
         end;
       end;
     end;
       CloseServiceHandle(schs);
   end;
    CloseServiceHandle(schm);
  end;
  Result := SERVICE_RUNNING = ss.dwCurrentState;
end;
```

Ancora un colpo messo a segno dagli hacker contro Microsoft

Internet Explorer KO!

Gli hackers realizzano una tripletta di exploit nel giro di un mese, in grado di mandare in crash il browser Internet Explorer. Un tre-a-zero secco che costringe Microsoft ad accelerare il rilascio delle ultime patch di sicurezza.

Girando in rete alla ricerca di nuovi bug e nuovi exploit da testare sulla mia macchina Windows, mi sono imbattuto in un sito molto curioso, aperto da poco tempo da un hacker che si fa chiamare *"Liu Die Yu"*, specializzato nello studio dei problemi di sicurezza nel mondo Windows. Il sito in questione è raggiungibile all'indirizzo *http://www.iebug.com* ed è una sorta di catalogatore automatico degli advisory di sicurezza, dei bug e degli exploit scoperti ogni giorno in merito ai seguenti prodotti Microsoft: Internet Explorer, Outlook e Windows Media. In sostanza, esiste una sorta di robot, programmato dal creatore del sito, che scansiona periodicamente le mailing list di sicurezza cercando qualsiasi bollettino riguardante i prodotti Microsoft citati, per poi segnarlo e catalogarlo sulla pagina indicata. Proprio spulciando tra le numerose righe apparse di recente su *IEBUG.COM*, ho selezionato tre diversi bug di Internet Explorer, in grado di mandare in crash (in maniera molto elementare) il browser di Microsoft con errori molto strani. Al momento si tratta di tre exploit in grado soltanto di crashare IE e causare qualche fastidio (anche perché l'attacco può essere integrato nei messaggi di posta tramite Outlook Express), ma non è escluso che in futuro qualcuno di questi exploit non possa tramutarsi in qualcosa di più pericoloso, ammesso che qualche hacker riesca a capire come controllare i registri di memoria sovrascritti in seguito al crash di Internet Explorer. Deviando dal consueto quindi, in questo numero di ioProgrammo studieremo questi tre diversi metodi che permettono di mandare crash IE, discutendo del contesto di applicazione degli exploit e degli effetti per ciascuno di questi.

Fig. 1: *Uno dei tanti crash di Internet Explorer che è possibile provocare usando gli exploit e i codici HTML descritti in queste pagine.*

L'EVENTO ONLOAD

Il primo dei tre bug riguarda il metodo *onLoad* presente nel linguaggio HTML ed è un errore riportato da un hacker chiamato *cipher* attraverso il suo sito *www.cipher.org.uk/index.php?p=cipher/advisories.cipher*. Chi mastica un pò di HTML, sa che all'interno del tag *<BODY>* di una pagina web è possibile inserire la proprietà *onLoad*, che permette di specificare uno script iniziale che verrà eseguito all'atto del caricamento della pagina. La combinazione del metodo *window.location* all'interno della proprietà *onLoad*, unita all'uso di alcuni codici speciali, riesce a crashare IE con un errore applicativo che forza la chiusura del browser e si sviluppa fino alla libreria di sistema *USER32.DLL*, passando attraverso *URLMON* e *MSHTML*. Il metodo *window.location* è uno dei tanti modi usati per specificare l'indirizzo di una pagina web e cambiare l'url corrente con quello di una nuova locazione; l'uso improprio di questo metodo causa non pochi problemi ad IE ed avviene quando invece di specificare un indirizzo del tipo *window.location = http://www.miosito.com*, si usa un link che punta ad un file locale, nella forma *window.location = file://c:\miofile*. Se al posto del drive *"c:\"* viene specificato un qualsiasi altro nome di drive utilizzando la codifica esadecimale (ad esempio *\xff:\ miofile*), Internet Explorer va in tilt e termina con un errore imprevisto nel tentativo di localizzare il file su un drive imprecisato.

```
<html>
<body onLoad=javascript:window.location=
                              "file://\xff:\autoexec.bat";>
IE CRASH-TEST NR 1 (onLoad)
</html>
```

Analizzando l'errore a livello di debug, fino a scendere nel codice assembly delle librerie crashate, si può notare come, a seguito del crash, vengano sovrascritti i tre registri *ECX*, *EDX* e *EDI*. Lo scopritore di questa vulnerabilità fa notare che tramite il nome di drive malformato è possibile controllare i primi 16 bit di *EDX* e *EDI*, esiste quindi una remota possibilità di tramutare questo crash in qualcosa di più pericoloso, come l'esecuzione di codice remoto (è necessario però approfondire e confermare questa ipotesi). In questi casi il problema più grosso per gli hacker è trovare un modo per pilotare l'esecuzione delle istruzioni e iniettare nello stack dove avviene il crash le proprie istruzioni che mirano a forzare la sicurezza del sistema. Vista la natura dell'exploit, realizzabile con una semplice riga del tag *<BODY>*, può essere usato in qualsiasi render HTML che faccia uso delle librerie di IE, come ad esempio MSN Messenger oppure Outlook Express.

UN PICCOLO KILLER IN JAVASCRIPT

Il secondo bug di IE presentato è una scoperta di Mike Mauler ed è anch'esso un errore abbastanza strano da osservare. Nel messaggio *http://seclists.org/ lists/fulldisclosure/2004/May/0757.html* postato sulla mailing list *full-disclosure,* viene osservato lo strano comportamento del browser Microsoft in presenza del seguente Javascript:

```
<html>
<script type="text/javascript">
        Wnd = window.createPopup();
        Wnd.document.body.innerHTML='<meta
                http-equiv="imagetoolbar" content="no">';
</script>
IE CRASH-TEST NR 2 (Javascript)
</html>
```

L'oggetto coinvolto è il solito *"popup"* di IE, che manda in crash il browser nel momento in cui lo script prova ad usare la proprietà *.innerHTML* (già da diverso tempo oggetto di molte altre vulnerabilità, ndr) nel modo illustrato poc'anzi. Anche in questo caso, come nell'exploit precedente, IE termina in modo imprevisto e va in crash con un errore applicativo nella libreria *MSHTML. DLL.* A differenza del bug precedente però, sembra che in questo caso non sia possibile prendere il controllo di alcun registro, perché l'errore è dovuto ad un banale *null-pointer* presente nel codice di IE. Da un lato questo ci rassicura e ci fa pensare che anche i programmatori di Microsoft devono combattere con i puntatori nulli (con scarsi risultati!) nel loro lavoro quotidiano.

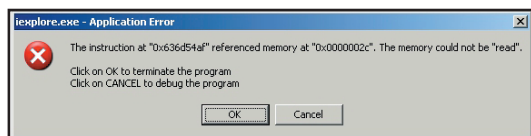



Fig. 2: **Il crash causato dall'exploit in Javascript è dovuto ad una referenza incosistente (null-pointer) presente nel codice del browser di Microsoft.**

UNO "STILE" CHE PUÒ COSTARE CARO!

Infine l'ultimo bug da segnalare interessa i fogli di stile, ovvero i famosi file *.CSS,* utilizzati dai web editor per uniformare l'aspetto delle pagine web e migliorare la visibilità di font e colori. Lo scopritore di quest'ultimo *"caso di crash"* ha pubblicato una demo del codice incriminato, testabile all'indirizzo *http://www.zeepost.nl/ ~henkie/index.html.* In questo caso l'errore scaturisce dall'uso dei fogli di stile in un punto non usuale delle pagine HTML: sembra che la chiamata ad un CSS inserita in prossimità dei tag di tabelle e di form (*<TABLE>* e *<FORM>*), provochi il crash indesiderato, dovuto quasi sicuramente ad un errore di parsing della libreria *MSHTML.DLL.* La pagina web che causa il crash di IE è la seguente:

```
<table>
<td>
<form class="quick">
<td>
</form>
</table>
<link rel="stylesheet" href=
                "http://www.zeepost.nl/~henkie/link.css">
```

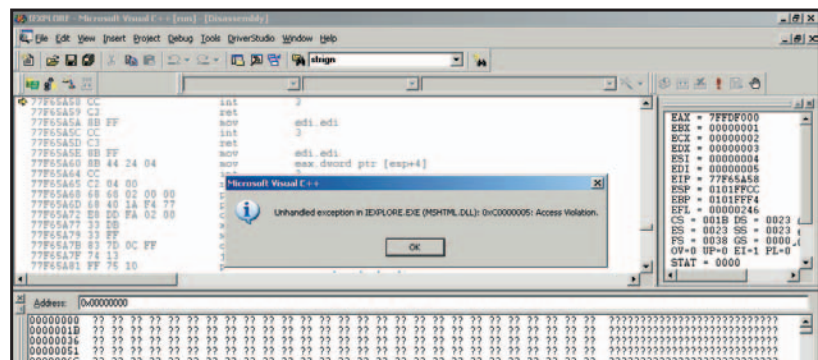



Fig. 3: **Seguendo il crash in modalità debug è possibile discendere nei meandri del codice assembly di IE, scoprendo magari se è possibile controllare i registri del processore e il flusso delle isruzioni.**

La parte critica del codice è quella riferita al tag *<LINK>*, dove si richiama il foglio di stile *LINK .CSS,* che deve risultare esterno rispetto alla pagina e contiene una semplice riga *.quick {float: left;}.* Nell'esempio abbiamo utilizzato un collegamento che punta allo stesso foglio di stile usato dallo scopritore della vulnerbabilità. La cosa più allarmante di quest'ultimo bug è il fatto che non dipende da script malformati, di conseguenza mentre per i primi due exploit è possibile limitare i danni attivando le restrizioni massime di sicurezza di IE (quelle che disabilitano gli script, per capirci), in quest'ultimo caso non si può fare a meno di subire il crash. Ancora non è stata condotta nessuna indagine sul bug in questione e non si è studiata la possibilità di sfruttare l'errore per eseguire codice remoto.

CONCLUSIONI

Abbiamo presentato tre exploit per Internet Explorer, di cui ancora non esiste una patch ufficiale e che funzionano con successo perfino sulle ultimissime versioni del browser, anche quelle dotate del *ServicePack1* e con tutte le patch di sicurezza del momento. E' d'obbligo sottolineare il fatto che i tre exploit possono essere ricondotti facilmente anche in altre applicazioni Microsoft che fanno uso delle stesse librerie di parsing HTML di IE, come ad esempio Outlook Express, MSN Messenger,e Windows Media Player. A titolo di prova abbiamo provato a realizzare una mail malformata, che integrando uno degli exploit presentati, è in grado di crashare Outlook Express nel momento in cui viene ricevuta e visualizzata.

Elio Florio

Costruiamo un automa capace di muoversi

Un robot che corre

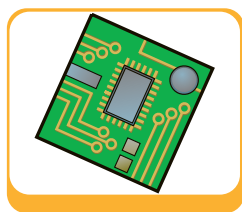
La realizzazione di un dispositivo meccanico non è né complessa, né costosa. Vediamo come sia possibile realizzarlo in un'ora di tempo e con meno di venti euro.

Verso la metà degli anni settanta, nell'industria, accadde un evento che condizionò decisamente le strategie produttive di ogni ciclo di assemblaggio: in quel periodo avvenne il sorpasso tra il costo orario di un addetto e quello di un robot industriale. Andando oltre le considerazioni di carattere sociologico nel mondo industrializzato, questo evento ha portato ad una progressiva maggiore diffusione dei robot industriali. Lungi dal volere descrivere le caratteristiche di una di queste macchine, lo scopo che ci prefiggiamo in questa sede è di costruire un semplice robottino mobile, che emulando i fratelli maggiori, utilizzati per trasportare pesi di quintali e talvolta tonnellate, deambulano in modo preciso ed ordinato in molte industrie. Il nostro robottino avrà in ogni modo caratteristiche meccaniche di tutto rispetto, sarà in grado di trasportare pesi di alcuni kilogrammi, con una precisione di navigazione dell'1% , significando con questo che durante il suo movimento, l'errore di posizione massimo sarà pari all 1% della distanza percorsa. Volendo dare una definizione più accurata della realizzazione, possiamo affermare che intendiamo progettare e costruire una base mobile azionata da ruote secondo la tecnica della trazione differenziale *(differential drive)* a tre gradi di libertà. Le dimensioni del robottino potranno essere variate secondo le necessità dell'utente, noi abbiamo realizzato una base mobile di 20 cm di lato: il software presentato in questa sede permette comunque di gestire applicazioni di dimensioni differenti. Al termine di questo articolo sarà presentato un programma di gestione della base robotizzata, in grado di controllare qualunque tipo di robot mobile a trazione differenziale.
Un'ultima importante specifica, che spero faccia contenti i molti studenti che mi scrivono, è il costo di realizzazione, che non dovrà superare i venti euro.

Fig. 1: *La base robotizzata è in grado di trasportare un carico utile di alcuni kilogrammi.*

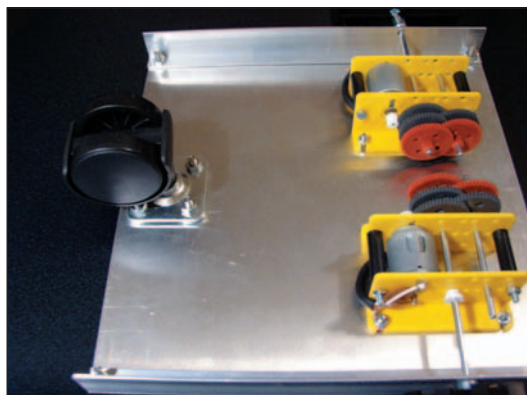



Fig. 2: *Questo mese la Elisys s.r.l. offre in omaggio i componenti per realizzare il robottino a chi acquista un sistema PC Explorer light.*

IL PRINCIPIO DI FUNZIONAMENTO

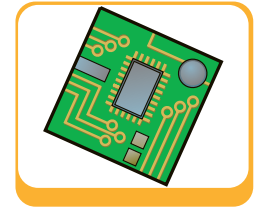
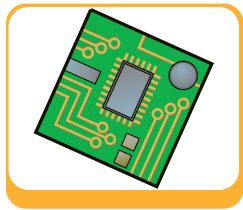
Il movimento del robot sul piano viene ottenuto per mezzo della trazione differenziale delle due ruote motrici. Per garantire il massimo della semplicità della applicazione, come stabilito nelle specifiche iniziali, si è deciso di rendere possibile l'avanzamento dei motori soltanto in avanti: l'implementazione del controllo avanti-indietro su ciascun asse può essere sviluppato successivamente, impiegando sistemi di controllo senz'altro più complessi di quelli che vengono proposti in questa sede. Il controllo della mobilità della macchina avviene quindi semplicemente controllando la commutazione fermo/ avanti di ciascun motore: quando entrambi i motori saranno fermi, il robot risulterà fermo, mentre quando entrambi ruotano in avanti la macchina procederà in avanti. Per ottenere la sterzata della nostra piccola realizzazione, è sufficiente azionare il motore corrispondente al lato opposto alla direzione nella quale si intende dirigere il robot. In altre parole, per fare ruotare il robot a destra, è sufficiente azionare il motore di sinistra mantenendo il destro fermo e viceversa. Il fulcro della rotazione è corrispondente al centro della ruota ferma, se potessimo fare ruotare i due motori in direzione opposta, sarebbe possibile

ottenere un movimento di rotazione del robot esattamente sulla posizione posta a metà strada tra i due pneumatici. A questo punto sarei tentato di inserire in queste pagine, una sfilza di integrali e di complessi calcoli matriciali corrispondenti allo sviluppo dei problemi di cinematica diretta ed inversa del moto di questo robot! Daremo invece alcuni concetti cardine di cinematica dei robot mobili, rimandando il lettore interessato ad eventuali approfondimenti alla lettura dei testi citati in bibliografia. Volendo modellizzare il comportamento cinematico di un robot mobile, possiamo dire innanzi tutto che la sua posizione istantanea può essere definita semplicemente per mezzo di tre variabili: ci troviamo di fronte quindi ad un sistema dotato di tre gradi di libertà. Lo stato della macchina può essere individuato dalla sua posizione rispetto ad un ipotetico sistema di riferimento, ad esempio cartesiano, e dal proprio orientamento. Supponiamo di desiderare che la macchina si muova da un punto di partenza 'A' e che per mezzo di opportuni comandi raggiunga un punto di arrivo 'D', variando il proprio orientamento secondo il percorso descritto nella figura precedente: desideriamo conoscere le coordinate cartesiane e l'orientamento finale del robot. Il processo descritto fino a questo punto riguarda un problema di cinematica diretta, risolvibile integrando le formule riportate in Fig. 4. Nel caso in cui desiderassimo invece raggiungere la posizione e l'orientamento corrispondenti alla posizione 'D', partendo dal punto 'A', ci troveremo di fronte ad un problema di cinematica inversa, nel quale occorre definire una strategia di ricerca del percorso che il robot deve mettere in atto fino al raggiungimento del 'goal' desiderato. In questa sede faremo soltanto un esempio di risoluzione del problema rappresentato in figura, poiché un algoritmo di ricerca generale dei percorsi risulta abbastanza complesso da sviluppare, rinviandone la trattazione ad un eventuale appuntamento futuro.
L'ambiente esterno, dotato di ostacoli e vincoli (oggetti fisici, dislivelli che impediscono il movimento della macchina od altri mezzi in movimento), spesso complica o rende talvolta impossibile la definizione del percorso da intraprendere per il raggiungimento della posizione desiderata. La risoluzione di un problema di cinematica inversa, giacché presuppone l'interazione del robot, dotato di limitazioni di movimento, con il mondo esterno, presuppongono la presenza di sensori esterni che consentano di rilevare la presenza degli ostacoli al movimento del robot. Ritornando al nostro esempio, possiamo definire l'algoritmo di risoluzione del problema in questione calcolando innanzitutto le circonferenze relative alla rotazione del robot in prossimità del punto di partenza 'A' e di quello di destinazione 'D'. Il calcolo della tangente alle due circonferenze fornisce la componente rettilinea del percorso da intraprendere. In altre parole è sufficiente far ruotare il robot, verso destra di una quantità pari all'angolo sotteso dall'arco di circonferenza A-B, procedere all'avanzamento in modo rettilineo fino al punto C, quindi fare ruotare nuovamente la macchina verso destra di un angolo sotteso dall'arco C-D fino al raggiungimento della posizione e dell'orientamento desiderato, corrispondente alla posizione di destinazione 'D'. La navigazione di precisione di una macchina industriale necessita tuttavia di una serie di sensori che rilevino in modo adeguato l'effettivo spazio percorso: nel nostro caso, provvederemo ad una taratura iniziale di alcune costanti caratteristiche del nostro sistema, che ci consentiranno di operare una 'previsione' della traiettoria percorsa dal robot con una ragionevole precisione.

Fig. 3: *I Robot mobili ad azionamento differenziale (Differential Drive), hanno il vantaggio della semplicità di realizzazione e controllo.*

Fig. 4: *La figura mostra le relazioni matematiche che legano la posizione del Robot alle velocità di rotazione dei pneumatici.*

LA REALIZZAZIONE MECCANICA

L'obiettivo principale della realizzazione meccanica è ovviamente quello di garantire una buona capacità motoria del nostro robot. Per mantenere la promessa fatta a molti lettori si è voluto garantire la massima reperibilità dei materiali ed un costo complessivo molto contenuto: tutto il materiale citato in questo paragrafo è reperibile presso il sito *www.opitec.it*. La base del robot è stata realizzata con un quadrato di lamiera di alluminio di 20x20 cm, dotato di uno spessore di 1 mm (Codice Opitec 806.530), ma qualunque supporto di dimensioni analoghe, anche in materiale plastico può andare bene lo stesso. I motori che garantiscono il movimento alla macchina sono in corrente continua, a bassa tensione (1,5-4,5V) e sono dotati di riduttore di giri per aumentarne la

Fig. 5: *I motori utilizzati sono in corrente continua a basso voltaggio e dotati di ingranaggi riduttori di velocità.*

L'apparecchiatura PC Explorer light è prodotta e commercializzata dalla Elisys s.r.l. e può essere acquistata al prezzo di € 213,60 nella versione light, € 99 nella versione basic e € 69 in kit (IVA inclusa) sul web all'indirizzo www.pcexplorer.it oppure inviando una e-mail all'indirizzo pcexplorer@elisys.it o anche telefonicamente al numero 0823/468565 o via Fax al: 0823/495483

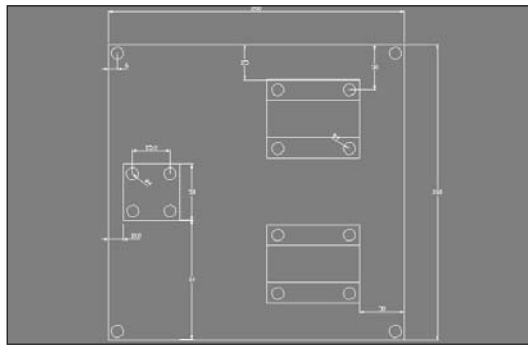



Fig. 6: Il progetto meccanico del Robot è incluso nel file 'Base_Robot.zip'.

coppia disponibile e diminuirne la velocità di rotazione. Il rapporto di riduzione dei motori può essere variato semplicemente sostituendo opportunamente i vari ingranaggi del riduttore per ottenere rapporti di riduzione compresi tra 5:1 e 3125:1, che assicurano una velocità di rotazione dell'asse del motore compresa tra circa 6 e 4000 giri al minuto. La trazione è assicurata da due pneumatici in gomma del diametro di 79mm, mentre un doppio ruotino ha lo scopo di garantire il terzo punto di appoggio del piano del robot. Il progetto meccanico è incluso nel file allegato *(Base_Robot.zip)* con il nome *'Progetto_Meccanico.JPG'.*

Quantità	Componente (www.opitec.it)	Codice Opitec
1	Lamiera 200x200	806.530
2	Motorino con ingranaggi e riduzione 3125:1	244.105
2	Pneumatico 79 mm	844.158
1	Ruotino posteriore	108.272
2	Profilati Alluminio 200 x 20 x 10 mm	
4	Fermacavo per biciclette	
8	Viti Diametro 3mm x 10 con dado	
8	Viti Diametro 4mm x 16 con dado	

TABELLA 1: Elenco dei componenti meccanici necessari alla realizzazione della base del robot.

ANALISI DELLO SCHEMA ELETTRICO

Il circuito elettronico che andiamo ad analizzare ha lo scopo di garantire il controllo dei due motori a corrente continua che azionano la base robotizzata.
Per rispondere alla legittima preoccupazione di molti lettori, in merito alla protezione elettrica del proprio PC, si è ritenuto opportuno utilizzare due accoppiatori ottici che assicurano una protezione ed un disaccoppiamento ottico tra il computer di controllo e la alimentazione elettrica dei motori. Per garantire la massima sicurezza della realizzazione si è provveduto a separare anche le masse elettriche delle due sezioni del circuito elettronico. Il controllo avviene per mezzo della porta seriale, della quale si sfruttano le due linee di controllo DTR e RTS. Dal punto di vista concettuale il circuito è molto semplice: quando le due linee di controllo si trovano a livello logico High, corrispondente ad una tensione compresa tra 3,5 e 25V a seconda del tipo di porta seriale, provvedono a fare illuminare il corrispondente diodo LED posto all'interno del relativo fotoaccoppiatore attraverso una resistenza di carico da 1 KOhm. Il fototransistor incapsulato nell'accoppiatore ottico 4N25, illuminato dalla luce proveniente dal LED, si porta in uno stato di conduzione, permettendo il flusso di corrente elettrica verso la base del transistor di potenza TIP112 che a sua volta permette l'alimentazione del motore elettrico. Il motore inizia così ad azionarsi, permettendo il movimento della base del robot. Le due linee DTR e RTS vengono collegate a due diodi LED della apparecchiatura PCExplorer light, che permettono la visualizzazione del relativo stato logico. Sul lato destro dello schema si possono notare le connessioni alle linee relative alla porta seriale e di alimentazione dell'apparecchiatura PC Explorer light. È opportuno notare che in questa configurazione elettrica, la componente logica, (collegata al PC) e la parte di potenza (connessa ai motori del robot), sono isolate elettricamente tra di loro, garantendo la massima sicurezza del nostro Personal Computer. Nel caso dell'accoppiatore ottico 4N25, la protezione elettrica può raggiungere una differenza di potenziale di migliaia di volts.

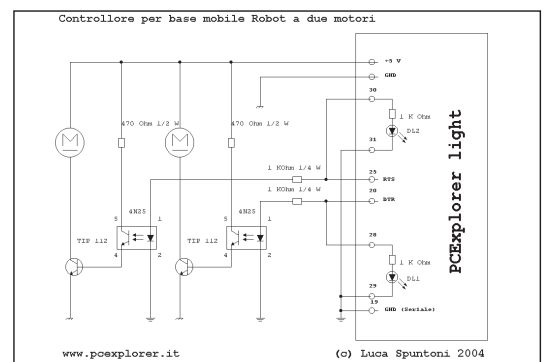



Fig. 7: In figura viene riportato lo schema elettrico del circuito di controllo, che per comodità e su richiesta dei lettori è stato inserito all'interno del file: 'Base_Robot.zip'.

Quantità	Componente (www.rs-components.it)	Codice RS
2	Optoisolatore 4N25	597-289
2	Transistor TIP 112	436-9779
2	Resistenza 1000 Ohm ? W	135-847
2	Resistenza 470 Ohm ? W	132-416
2	Cavi 2 poli + calza	

TABELLA 2: Elenco dei componenti elettronici necessari alla realizzazione del circuito di controllo.

REALIZZAZIONE DEL CIRCUITO ELETTRONICO

Il circuito elettronico per l'azionamento della base robotizzata può essere realizzato senza saldature

per mezzo della apparecchiatura PC Explorer light, ma anche con i soli componenti elettronici elencati a lato di queste pagine, per mezzo del loro montaggio su una comune basetta millefori, utilizzando un comune saldatore a stagno. Il lettore potrà reperire i pochi componenti elettronici che utilizzeremo per la costruzione della apparecchiatura, in qualunque negozio di componenti elettronici, oppure per corrispondenza presso la RS Components *www.rs-components.it*; l'assemblaggio del circuito può essere effettuato senza saldature per mezzo dell'apparecchiatura PCExplorer light.

IL SOFTWARE C++

Il software di controllo della nostra applicazione permette di risolvere il problema della determinazione della posizione e dell'orientamento della nostra base robotizzata che, come abbiamo detto in precedenza, riguarda una applicazione di cinematica diretta. Il codice sorgente, scritto in C++ viene messo a completa disposizione del lettore ed è disponibile nel CD con il nome *'Base_Robot.zip'*. In questa sede analizziamo soltanto le parti più significative, rimanendo a disposizione del lettore per ogni chiarimento all'indirizzo: *luca.spuntoni@ioprogrammo.it*. L'intestazione del programma evidenzia i componenti utilizzati, nonché la dichiarazione delle variabili globali riguardanti le coordinate assolute del Robot, espresse in millimetri dal punto di origine sul piano di lavoro ed in gradi sessangesimali.

```
#include <vcl.h>
#pragma hdrstop
#include "SpuntoBaseRobotUnit.h"
#include <math.h>
//——————————————————————————
#pragma package(smart_init)
#pragma link "SpuntoLedComponent"
#pragma link "TSpuntoHardwarePortIO_unit"
#pragma link "CSPIN"
#pragma resource "*.dfm" //Communication parameters
unsigned short datain, CombaseAddress, MCRAddress,
                    LCRAddress, MSRAddress;
Extended XCoord, YCoord, Theta; //Robot Global
          Position (millimeters,millimeters,Degrees)
```

La procedura *FormCreate* provvede ad impostare le variabili globali del programma, oltre che a predisporre alcuni componenti della Form. Notiamo in particolare l'azzeramento delle variabili corrispondenti ai tre gradi di libertà dell'apparecchiatura *(Xcoord, Ycoord, Theta)*, nonché l'impostazione degli indirizzi fisici corrispondenti alla porta di comunicazione seriale.

```
void __fastcall TSpuntoBaseRobotForm::FormCreate(
                                   TObject *Sender)
{  // Sets the Initial Coordinates
  XCoord=0;
  YCoord=0;
  Theta=0;
  // Sets the Power LED
  if(PowerONSpeedButton->Down)
    SpuntoPowerLed->LedOn();
  else
    SpuntoPowerLed->LedOff();
  // Both Motor Buttons Up
    LeftMotorFWD->Down=False;
    RightMotorFWD->Down=False;
  // Default (COM1) Port setup
    CombaseAddress=0x03f8;
    MCRAddress=CombaseAddress+4;
    LCRAddress=CombaseAddress+3;
    MSRAddress=CombaseAddress+6; }
```

Il cuore del programma risiede nella funzione *MainTimerTimer*, chiamata dal timer *MainTimer* una volta al secondo. Vengono definite innanzitutto due variabili locali che hanno lo scopo di contenere il valore di velocità, espresso in millimetri al secondo del pneumatico sinistro e di quello destro *(LeftComp, RightComp)*.

Fig. 8: L'immagine mostra un dettaglio del posizionamento dei componenti elettronici.

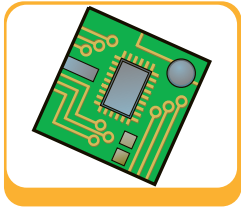
```
void __fastcall TSpuntoBaseRobotForm::MainTimerTimer(
                                   TObject *Sender)
{ //Main Timer
  Extended LeftComp, RightComp; //Left and Right
                              movement components
```

Nel caso in cui il pulsante di alimentazione sia attivato, viene acceso il LED corrispondente sulla form e si provvede a leggere lo stato delle linee della porta seriale. A questo punto vengono verificate le posizioni dei pulsanti corrispondenti allo stato di marcia dei motori sinistro e destro ed in caso d'attivazione di questi componenti, si provvede a modificare la linea elettrica corrispondente a ciascun motore, in modo da forzarla a livello logico *HIGH*. In particolare per azionare il motore di sinistra, si opera l'OR logico tra il valore presente sulla porta ed il valore esadecimale *02h* (l'OR logico di qualsiasi valore con 00000010 binario corrispon-

Fig. 9: In figura è visibile un dettaglio della connessione dei cavi dei motori del robot.

• ROBOTICA (Zanichelli) 2003
• ROBOTICA INDUSTRIALE (Mc Graw-Hill) 2000
• MODELLISTICA DEI ROBOT INDUSTRIALI (Celid) 2002

PRECAUZIONI

Prima di collegare il circuito al nostro PC occorre verificare la nostra realizzazione con attenzione per assicurarci che tutto sia stato collegato come previsto.

de a forzare il bit *n1* ad '1'). Per arrestare il motore di sinistra si opera l'*AND* logico del valore della porta con *FDh*, in modo da forzare il bit *n1* a '0'. In modo analogo per la gestione del motore destro viene operato l'*OR* logico del valore della porta con *01h* per azionarlo e l'*AND* con *Feh* per arrestarlo, operazioni che modificano entrambe il bit *n0* del registro *MCR* della porta seriale. Terminata la definizione dello stato logico dei primi due bit del registro *MCR*, si opera la scrittura del byte ottenuto per mezzo dell'istruzione *SpuntoHardwarePort->WritePort(MCRAddress, datain)*.

```
if (PowerONSpeedButton->Down)  //Power is ON
{  SpuntoHardwarePort->LedOn();
   datain=SpuntoHardwarePort->ReadPort(MCRAddress);
   DelayTimer->Enabled=true;
    if (LeftMotorFWD->Down==True) // Left Motor On
    { datain=(datain | 0x02); }
    else   // Left Motor Off
    { datain=(datain & 0xfd); }
    if (RightMotorFWD->Down==True) // Right Motor On
    { datain=(datain | 0x01); }
    else       // Right Motor Off
    { datain=(datain & 0xfe); }
    SpuntoHardwarePort->WritePort(MCRAddress,datain); }
else
  { SpuntoHardwarePort->LedOff();   // Power is OFF
  datain=SpuntoHardwarePort->ReadPort(MCRAddress);
  datain=(datain & 0xfc);         //DTR and RTS '0'
  SpuntoHardwarePort->WritePort(MCRAddress,datain);
  DelayTimer->Enabled=false;
  LeftMotorFWD->Down=False;   // Both Motor Buttons Up
  RightMotorFWD->Down=False;
  }
```

Fig. 10: ***La precisione direzionale del robot è favorita dalla notevole distanza presente tra i due pneumatici dotati di trazione.***

La risoluzione del problema cinematico diretto di calcolo delle coordinate e dell'orientamento del Robot avviene calcolando le velocità di avanzamento di ciascun pneumatico, acquisendo direttamente i parametri fisici del diametro dei pneumatici e di velocità di rotazione dai componenti *LeftWheelDiameterSpinEdit* e *LeftTimeConstantSpinEdit:* quest'ultimo parametro in particolare viene ricavato misurando il tempo impiegato da ciascuna ruota per compiere dieci giri completi.

```
    if (LeftMotorFWD->Down==True) // Left Motor On
    { LeftComp=mypi*LeftWheelDiameterSpinEdit->Value/(
                    LeftTimeConstantSpinEdit->Value/10); }
    else
    { LeftComp=0; // Left Motor Off }
    if (RightMotorFWD->Down==True) // Right Motor On
    { RightComp=mypi*RightWheelDiameterSpinEdit->Value/(
                    RightTimeConstantSpinEdit->Value/10);}
    else
    { RightComp=0; // Right Motor Off}
```

Il primo parametro cinematico ad essere calcolato è l'angolo *Theta*, che definisce l'orientamento della base robotizzata rispetto alla direzione di partenza, calcolato in gradi sessangesimali e positivo quando la rotazione è antioraria. È possibile riconoscere la formula descritta in precedenza, per la risoluzione della quale è operata un'integrazione ad intervalli prefissati per default di un secondo. In modo analogo vengono calcolate le coordinate assolute del Robot: è degna di nota la conversione da gradi sessangesimali a radianti (1 *Radiante= 180/π Gradi*) che viene operata all'interno delle funzioni relative al calcolo delle coordinate *X* e *Y*.

```
// Theta Calculation
  Theta= Theta+(0.5*(RightComp-LeftComp)*(
                        MainTimer->Interval/1000));
// X Coordinate calculation
  XCoord=XCoord+(0.5*(LeftComp+RightComp)*(cos(
      Theta*(mypi/180)))*(MainTimer->Interval/1000));
  // Y Coordinate calculation
 YCoord=YCoord+(0.5*(LeftComp+RightComp) *(sin(
      Theta*(mypi/180)))*(MainTimer->Interval/1000));
```

Qualora l'orientamento della apparecchiatura ecceda dai limiti 0-360°, si provvede a riportare il valore dell'angolo entro questi valori.

```
//Theta angle correction
  if (Theta>360) Theta=(Theta-360);
  if (Theta<0) Theta=(Theta+360);
```

Al termine della fase di calcolo vengono impostate le etichette della form che provvedono a visualizzare i dati cinematici della nostra base robotizzata.

```
//Sets the labels
  ThetaStaticText->Caption=FloatToStr(Theta);
  XcoordStaticText->Caption=FloatToStr(XCoord);
  YcoordStaticText->Caption=FloatToStr(YCoord);
  Leftwheelspeed->Caption=FloatToStr(LeftComp);
  Rightwheelspeed->Caption=FloatToStr(RightComp); }
```

ESECUZIONE ED INSTALLAZIONE DEL PROGRAMMA

L'installazione e l'esecuzione del programma non

crea difficoltà, il software viene fornito anche nella versione completamente compilata e collaudata. Per garantire una miglior portabilità con le versioni successive il programma è stato sviluppato con Borland C++ Builder 4. I programmatori che utilizzano VC++ non dovrebbero comunque riscontrare problemi nel caso di utilizzo dei codici sorgenti in questo tipo di ambiente. Tutti i file necessari all'utilizzo del programma sono presenti in forma compilata e dotati di codice sorgente all'interno del file: *Base_Robot.zip*. Sono stati utilizzati due componenti sviluppati con Delphi *(TspuntoHardwarePortIO* e *SpuntoLedComponent)* che vengono distribuiti nel file allegato al CD della rivista in forma compilata e dotati dei relativi files '.hpp': chiunque desideri ricevere il codice sorgente di questi componenti può richiederli all'indirizzo *spuntosoft@tiscali.it*. L'utilizzo del programma su sistemi dotati di Win 2000, XP oppure NT, è possibile seguendo le istruzioni riportate di seguito. Al fine di consentire l'esecuzione del software, poiché questo accede direttamente all'hardware della macchina, è necessario installare il driver: *'PortTalk - A Windows NT/ 2000/XP I/O Port Device Driver Version 2.2'* scaricabile all'indirizzo: *'http:// www.beyondlogic.org/porttalk/porttalk.htm'*. Il file *'Porttalk22.zip'* contiene tutte le istruzioni necessarie all'utilizzo corretto del driver, nonché una notevole mole di informazioni relative all'accesso delle porte hardware del PC: viene fornito inoltre il codice sorgente completo delle applicazioni. Per quanto riguarda l'utilizzo del programma proposto in questa sede sotto Win 2000/NT/XP è sufficiente estrarre i file contenenti il driver 'Porttalk' nella directory del programma da utilizzare e 'chiamare' l'applicazione *'SpuntoBaseRobot.exe'* per mezzo del comando: *'C:\> Allowio SpuntoBaseRobot.exe/ a'* che consente l'accesso da parte dell'eseguibile *'SpuntoBaseRobot.exe'* a tutte le porte del PC. Lanciando il programma si ottiene una unica finestra contenente pochi comandi dall'utilizzo abbastanza intuitivo. La prima operazione da compiere riguarda l'impostazione delle costanti di tempo di ogni motore, ricavata semplicemente misurando il tempo impiegato da ciascun pneumatico a compiere dieci rotazioni complete: nel nostro caso abbiamo inserito 125 secondi per ciascun asse. Occorre impostare successivamente il diametro di ciascun pneumatico: questa applicazione può infatti gestire anche robot asimmetrici, ad esempio perché dotati di ruote di differenti dimensioni in caso di applicazioni speciali. Selezioniamo, a questo punto, la porta seriale sulla quale è collegata l'applicazione e premiamo il pulsante *'ON / OFF'*. Premendo i pulsanti di attivazione dei motori *(Left Motor e Right Motor)*, possiamo alimentare i rispettivi azionatori, notando che il programma inizia a calcolare le coordinate relative alla posizione del robot, misurate dal punto di partenza ed espresse in millimetri e gradi sessangesimali. È possibile azzerare le coordinate del robot premendo il pulsante *'Reset Robot Coordinates'*. A questo punto la applicazione è completa ed è possibile verificare il funzionamento della nostra base robotizzata: personalmente ho verificato che la precisione nei calcoli della posizione e dell'orientamento del sistema è contenuto entro i limiti prefissati dell'1% di errore, trasportando un carico utile di circa tre kilogrammi, più che sufficiente per trasportare la colazione dalla cucina alla camera da letto!

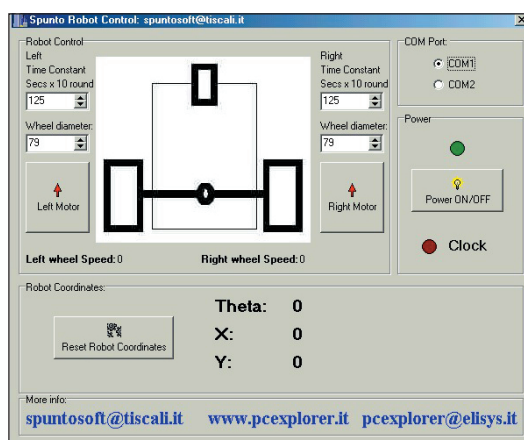



Fig. 11: *Il software descritto in queste pagine è presente nel CD allegato alla rivista e sul sito web www.ioprogrammo.it completo di codice sorgente (Base_Robot.zip).*

CONCLUSIONI

Nonostante il limitato spazio a disposizione, abbiamo visto come realizzare una base robotizzata completa: il progetto dello schema elettrico e meccanico, tutti i collegamenti necessari, il software compilato ed i relativi codici sorgenti sono stati messi a completa disposizione del lettore. Il lettore vorrà comprendere che, nonostante quanto esposto in queste pagine ne sia stato debitamente verificato e collaudato, tuttavia viene riportato a scopo illustrativo e di studio, pertanto l'editore e l'autore non sono da considerare responsabili per eventuali conseguenze derivanti dall'utilizzo di quanto esposto in questa sede, soprattutto per la tipologia e la complessità dell'argomento.

Luca Spuntoni

Fig. 12: *Il programma è in grado di calcolare con precisione la posizione del Robot.*

CODICE ALLEGATO ALLA RIVISTA

Il codice sorgente della applicazione e tutti i componenti necessari sono reperibili sul CD allegato alla rivista all'interno del file: *'Base_Robot.zip'*.

L'autore è lieto di rispondere ai quesiti dei lettori sull'interfacciamento dei PC all'indirizzo:
luca.spuntoni@ioprogrammo.it

Sfruttare le funzioni XML avanzate di Office 2003

Fatture automatiche con Smart Document

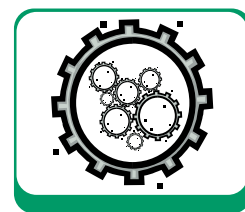
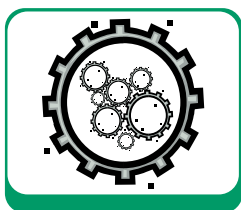
Analisi su come implementare uno Smart Document utile per la gestione dei dati di un documento contabile. In modo semplice la problematica XML su Office sarà risolta.

Gli Smart Document costituiscono l'innovazione più importante introdotta in Office 2003 Professional. Sono strumenti che uniscono i vantaggi delle applicazioni desktop con quelli della tecnologia XML. Gli Smart Document, per quanto riguarda l'intelligenza "contestuale", sono l'evoluzione degli Smart Tag e, come vedremo, non forniscono soltanto suggerimenti! Gli Smart Document sono creati con lo Smart Document Software Development Kit che può essere scaricato dal sito della Microsoft. In questo e nel successivo appuntamento descriveremo come sviluppare uno Smart Document e, come esempio, presenteremo una "fattura intelligente" che, durante la compilazione, in base alle scelte dell'utilizzatore, può gestire diversi scenari operativi. Prima di addentrarci nello sviluppo, parleremo delle parti principali dello Smart Document Software Development Kit. Gli esempi che presenteremo sono stati sviluppati e testati sulla piattaforma Office 2003 Professional – Windows XP. Per affrontare lo studio degli Smart Document bisogna avere delle conoscenze sull'XML e sugli Smart Tag per questo vi consigliamo di consultare i nostri precedenti articoli.

SMART DOCUMENT

Utilizzabili con Word 2003 ed Excel 2003, gli Smart Document sono strumenti, particolarmente utili per la creazione di soluzioni che si sviluppano secondo un determinato processo. Gli Smart Document, per esempio, sono utilizzabili per la creazione di documenti contabili, report speciali, documenti legali, contratti, articoli per giornali ecc; in buona sostanza, per tutti quei documenti che devono essere modificati, approvati, compilati seguendo un particolare percorso e soltanto da persone autorizzate. Per capirci, prendiamo in considerazione il caso della compilazione di un ordine di vendita: il venditore crea un ordine e lo passa al suo responsabile per l'approvazione. Quest'ultimo, dopo l'approvazione, lo passa al responsabile del magazzino per la verifica e la consegna della merce. In questo caso, se gestissimo il tutto con uno Smart Document, i vari passaggi del documento potrebbero essere attivati attraverso dei pulsanti, il riconoscimento degli utenti potrebbe avvenire automaticamente ed i dati dei clienti, degli articoli ecc. potrebbero essere recuperati da un database. Gli Smart Document possono essere implementati con vari linguaggi (Visual Basic 6, C#, C++ e VB Net), sia in tecnologia COM che .NET, e possono interagire con database Access e Sql-Server. È anche possibile inviare email attraverso Outlook o creare presentazioni in PowerPoint. Gli Smart Document possono essere sviluppati per una rete locale o geografica e quindi usare Web Service e Siti Web basati su SharePoint Portal Server 2003. Uno Smart Document completo, in generale, può comprendere differenti tipi di file, alcuni fondanti ed altri dipendenti dalle funzionalità implementate e da come sarà distribuito.

SMART DOCUMENT SDK

Lo Smart Document SDK è lo strumento da utilizzare per implementare uno Smart Document. Esso è un ottimo supporto per lo sviluppo di Smart Document con le tecnologie COM e .NET. La guida dell'SDK è divisa in diverse sezioni che forniscono informazioni sulla pianificazione, sullo sviluppo, sulla distribuzione e sulla sicurezza degli Smart Document. Con l'SDK, inoltre, è fornito il tool *XMLSign*, per la certificazione degli XML Espansion Pack, e il file *"Disable XML Expansion Pack Manifest Security"* per la disattivazione del processo di verifica della certificazione dello Smart Document. Questo file è un file del registro di sistema, di una sola riga, che serve per cambiare il valore della DWord *"DisableManifestSecurityCheck"*.

IL PRIMO SMART DOCUMENT

Innanzitutto precisiamo che gli Smart Document forniscono suggerimenti, attraverso la scheda *Azioni Documenti*, sul *Riquadro Attività*. Quando si apre un documento, che ha associato un pacchetto di espansione XML, viene aperto automaticamente il riquadro *Azioni Documenti* con delle informazioni e gli strumenti per compiere delle azioni; le altre informazioni e gli altri strumenti, programmati nello Smart Document, verranno mostrate man mano che si selezioneranno gli elementi XML presenti sul documento. L'esempio che illustreremo in questo e nel successivo appuntamento è una *Smart Fattura* che si auto-struttura in base alle scelte fatte dall'utilizzatore. Quest'ultimo può scegliere i dati identificativi ed il logo dell'azienda, i dati del destinatario (il cliente), il tipo di pagamento, dei frammenti di testo (con i quali l'azienda fornisce delle informazioni)… ed altre cose che vedremo nel successivo appuntamento. Per illustrare la tecnologia alla base degli Smart Document, presenteremo, soltanto, come gestire i dati dei Clienti ed il logo aziendale. Questo ci permetterà di capire come amministrare il pannello *Azioni Documenti* (e di utilizzare uno schema XSD simile a *Persona.xsd* illustrato nei precedenti articoli).
Prima d'introdurre l'esempio presentiamo la *ISmartDocument*.

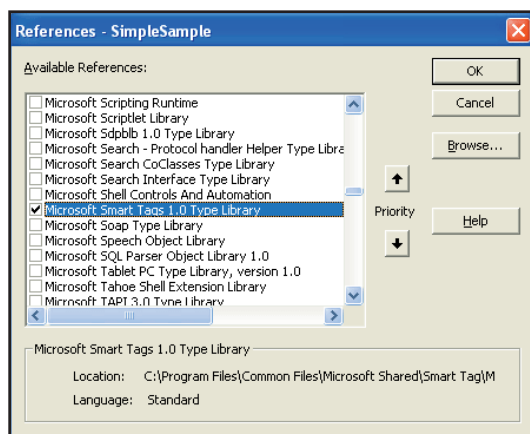



Fig. 1: *L'errore nel riferimento alla libreria degli Smart Document.*

Metodo	Breve descrizione
PopulateCheckbox	Specifica il valore iniziale di un check box
PopulateDocumentFragment	Utilizzato per specificare un frammento di testo da inserire nel riquadro azioni
PopulateHelpContent	Utilizzato per specificare un file di Help
PopulateImage	Specifica un'immagine da caricare
PopulateListOrComboContent	Utilizzato per specificare il contenuto di una listbox o di un combobox
PopulateTextboxContent	Specifica il contenuto di un textbox
SmartDocInitialize	Specifica l'azione da fare quando si inizializza lo smart document

TABELLA 1: Alcuni metodi della ISmartDocument.

NOTA

INTERFACCIA E IMPLEMENTS

Per implementare un'interfaccia, in un modulo di classe, bisogna anteporre, al nome dell'interfaccia, la parola chiave Implements. Questo comporta la creazione di tutti gli attributi e metodi pubblici dell'interfaccia nel modulo di classe. Ricordiamo che per "contratto" il modulo di classe dovrà implementare tutti gli elementi nell'interfaccia.

ISMARTDOCUMENT

L'interfaccia *ISmartDocument* fornisce l'accesso alla Smart Document API. L'interfaccia ha diverse proprietà e diversi metodi per amministrare i controlli (dello Smart Document) da mostrare sul riquadro *Azioni Documenti*.
Nelle *Tabelle 1* e *2* abbiamo riportato alcuni degli elementi della *ISmartDocument*. La *Tabella 3*, invece, contiene alcuni degli oggetti che possono essere gestiti sul riquadro *Azioni Documenti* (valori della proprietà *ControlTypeFromID*).

SMART CLIENTI

Per implementare, in Visual Basic 6, uno Smart Document che permette di gestire i dati dei clienti ed il logo aziendale, di un documento contabile, abbiamo bisogno, almeno, dei seguenti elementi.

1. Uno schema XML che definisce gli elementi sensibili al "contesto". Nel nostro caso gestiamo i seguenti elementi: *ragionesociale, indirizzo, città, tipo cliente* e *logo aziendale.*
2. Un progetto DLL Activex, che attraverso l'interfaccia *ISmartDocument*, permette di gestire gli oggetti associati agli elementi XML. Nel nostro caso dei textbox (per ragionesociale ed indirizzo), un ComboBox (per le città), una listbox (per tipo cliente) ed un controllo immagine (per il logo).
3. Un file *Manifest.xml* (ed altri elementi) per distribuire lo Smart Document, questo lo vedremo nel prossimo articolo.

LO SCHEMA

Ricordiamo che lo schema XSD serve per definire la struttura dei documenti XML. Nel nostro caso lo schema lo nominiamo *"clienteelogo.xsd"*. Di seguito riportiamo il codice dello schema.

```
<xsd:schema xmlns:xsd=
            "http://www.w3.org/2001/XMLSchema"
 xmlns="clienteelogo"
 targetNamespace="clienteelogo"
 elementFormDefault="qualified">
 <xsd:complexType name="clienteelogo">
   <xsd:all>
    <xsd:element name="ragionesociale" type="xsd:string" />
    <xsd:element name="indirizzo" type="xsd:string" />
    <xsd:element name="citta" type="xsd:string" />
    <xsd:element name="tipo" type="xsd:string" />
    <xsd:element name="logo" type="xsd:string" />
   </xsd:all>
 </xsd:complexType>
```

```
  <xsd:element name="clienteelogo" type="clienteelogo"/>
</xsd:schema>
```

Nello schema abbiamo previsto un *ComplexType* con l'elemento *"all"* (così nei file XML gli elementi non devono essere posti nell'ordine definito nello schema). Ricordiamo che lo schema ai documenti Word (Excel), può essere associato con la maschera "*Modelli e Aggiunte*". Dopo aver associato lo schema dovete creare un documento XML (Fig. 2). Ora descriviamo come implementare la DLL che associa degli oggetti (e delle azioni) agli elementi XML che avranno intelligenza "contestuale".

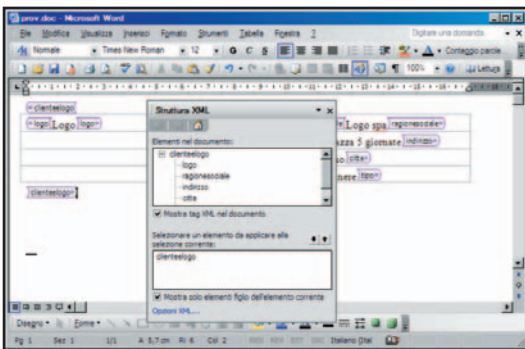



Fig. 2: **La struttura XML dell'esempio.**

IL PROGETTO

Create un progetto DLL Activex con un riferimento alla seguente libreria: *"Microsoft Smart Tag 2.0 Type Library"*. Attenzione! In alcuni casi, nella finestra dei riferimenti, è mostrato un riferimento alla prima versione della libreria cioè *"Microsoft Smart Tag 1.0 Type Library"* (Fig. 1), per risolvere il problema (dopo l'associazione) basta chiudere e riaprire il progetto. Vediamo le dichiarazioni da inserire nella classe del progetto.

```
Implements ISmartDocument
Const cNAMESPACE As String = "clienteelogo"
Const cRSOCIALE As String = cNAMESPACE &
                                          "#ragionesociale"
Const cINDIRIZZO As String = cNAMESPACE & "#indirizzo"
Const cCITTA As String = cNAMESPACE & "#citta"
Const cTIPO As String = cNAMESPACE & "#tipo"
Const cLOGO As String = cNAMESPACE & "#logo"
Const cTYPES As Integer = 5
Private strPath  As String
```

Con la prima istruzione si dichiara che nel modulo di classe verrà implementata l'interfaccia *ISmartDocument*. Le dichiarazioni delle costanti, invece, riguardano gli elementi dichiarati nel file *clienteelogo*. In particolare, dichiariamo una costante (nel formato *namespace#nomeelemento*) per ogni elemento definito nello schema (attenzione ai nomi, poiché l'XML è case-sensitive). La costante *cTYPES*, invece, rappresenta il numero di elementi che hanno associato un controllo, nel nostro caso 5 dato che prevediamo: due textbox, un combobox, una listbox e un controllo immagine. La variabile *strPath* conterrà il path del documento (o cartella Excel) come vedremo tra poco. Il primo metodo che consideriamo è *SmartDocInitialize*.

```
Private Sub ISmartDocument_SmartDocInitialize(..., _
                        ByVal Document As Object,... ,...)
  strPath = Document.Path & "\"
End Sub
```

Proprietà	Breve descrizione
ControlCaptionFromID	Specifica la caption del controllo
ControlCount	Specifica il numero di controlli associati ad un elemento XML
ControlID	Specifica l'ID di un controllo.
ControlNameFromID	Specifica una stringa che può essere usata per l'accesso ai controlli con VBA.
ControlTypeFromID	Specifica il tipo di controllo
SmartDocXmlTypeCaption	Specifica la caption per un gruppo di controlli
SmartDocXmlTypeCount	Specifica il numero di elementi che hanno associato azioni nello Smart Document
SmartDocXmlTypeName	Specifica il nome dell'elemento con associate un'azione

TABELLA 2: Le proprietà della ISmartDocument.

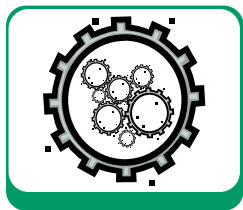
Valore ControlTypeFromID	Elemento specificato
C_TYPE_ACTIVEX	ActiveX control
C_TYPE_BUTTON	Command button.
C_TYPE_COMBO	Combo box
C_TYPE_DOCUMENTFRAGMENT	Frammento di documento
C_TYPE_HELP	Help
C_TYPE_IMAGE	Immagine
C_TYPE_LABEL	Label
C_TYPE_LINK	Link
C_TYPE_LISTBOX	List box
C_TYPE_TEXTBOX	TextBox

TABELLA 3: Alcuni controlli gestiti dagli Smart Document.

Nella procedura precedente impostiamo il valore della *strPath* sul path del documento che si sta creando. La *strPath* sarà considerata quando caricheremo le immagini per il logo. Notate che della procedura abbiamo riportato soltanto l'argomento utilizzato. Procediamo impostando il codice per i vari elementi XML associati ai controlli mostrati sul riquadro *Azioni Documenti*.

DAGLI ELEMENTI XML AI CONTROLLI

Definiamo il numero di elementi XML che hanno associato un controllo per far ciò usiamo la seguente.

```
Private Property Get ISmartDocument_
                     SmartDocXmlTypeCount() As Long
```

```
    ISmartDocument_SmartDocXmlTypeCount = cTYPES
End Property
```

Specifichiamo i nomi degli elementi, associati, alle azioni dello Smart Document, attraverso le costanti definite prima.

```
Private Property Get ISmartDocument SmartDocXmlTypeName
                                (ByVal XMLTypeID As Long) As String
    Select Case XMLTypeID
        Case 1
            ISmartDocument_SmartDocXmlTypeName =
                                                        cRSOCIALE
        Case 2
            ISmartDocument_SmartDocXmlTypeName =
                                                        cINDIRIZZO
        Case 3
            ISmartDocument_SmartDocXmlTypeName = cCITTA
        Case 4
            ISmartDocument_SmartDocXmlTypeName = cTIPO
        Case 5
            ISmartDocument_SmartDocXmlTypeName = cLOGO
        Case Else
    End Select
End Property
```

Specifichiamo le Caption che verranno mostrate sul riquadro *Azioni Documenti.*

```
Private Property Get ISmartDocument_
                            SmartDocXmlTypeCaption( _
                        ByVal XMLTypeID As Long,) As String
    Select Case XMLTypeID
        Case 1
            ISmartDocument_SmartDocXmlTypeCaption =
                                                "Ragione Sociale"
        Case 2
            ISmartDocument_SmartDocXmlTypeCaption =
                                                "Indirizzo"
        Case 3
            ISmartDocument_SmartDocXmlTypeCaption = "Città"
        Case 4
            ISmartDocument_SmartDocXmlTypeCaption =
                                                "Tipo Cliente"
        Case 5
            ISmartDocument_SmartDocXmlTypeCaption =
                                                "Seleziona il logo"
        Case Else
    End Select
End Property
```

Ora definiamo i controlli per ogni tipo definito in precedenza. Per fare ciò utilizziamo i seguenti elementi:

1. **ControlCount** per specificare il numero di controlli dello stesso tipo;
2. **ControlID** per assegnare un identificatore numerico, univoco, ai controlli;
3. **ControlNameFromID** per specificare il nome che potrà essere usato con VBA;
4. **ControlCapitionFromID** per specificare la caption del controllo, mostrata nel riquadro attività;
5. **ControlTypeFromID** per specificare il tipo di controllo.

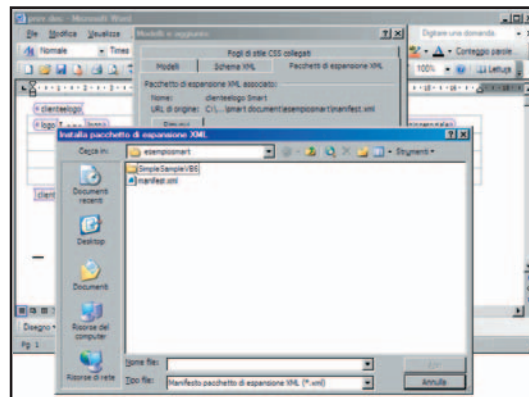



Fig. 3: ***Fase di caricamento del pacchetto clienteelogo Smart.***

```
Private Property Get ISmartDocument_ControlCount( _
                ByVal XMLTypeName As String) As Long
    Select Case XMLTypeName
        Case cRSOCIALE
            ISmartDocument_ControlCount = 1
            Case cINDIRIZZO
            ISmartDocument_ControlCount = 1
            Case cCITTA
            ISmartDocument_ControlCount = 1
            Case cTIPO
            ISmartDocument_ControlCount = 1
            Case cLOGO
            ISmartDocument_ControlCount = 2
            `inseriamo due immagini sulla scheda azioni
                                                documenti
        Case Else
    End Select
End Property
Private Property Get ISmartDocument_ControlID( _
                ByVal XMLTypeName As String, _
            ByVal ControlIndex As Long) As Long
    Select Case XMLTypeName
        Case cRSOCIALE
            ISmartDocument_ControlID = ControlIndex
        Case cINDIRIZZO
            ISmartDocument_ControlID = ControlIndex + 100
        Case cCITTA
            ISmartDocument_ControlID = ControlIndex + 200
        Case cTIPO
            ISmartDocument_ControlID = ControlIndex + 300
        Case cLOGO
            ISmartDocument_ControlID = ControlIndex + 400
    End Select
End Property
```

ISMARTDOC-PROPERTIES

Questa interfaccia permette di amministrare le proprietà principali dello Smart Document quali posizione dei controlli nel riquadro azioni documenti, caratteristiche dei controlli che contengono del testo, caratteristiche dei controlli con più righe (combobox, listbox) o che possono contenere password (textbox), caratteristiche delle immagini, caratteristiche dei frammenti di documenti o di help e dei controlli Activex mostrati sul riquadro azioni documenti.

```
Private Property Get ISmartDocument_ControlNameFromID
                         (ByVal ControlID As Long) As String
    ISmartDocument_ControlNameFromID =
                                  cNAMESPACE & ControlID
End Property
Private Property Get ISmartDocument_
                                   ControlCaptionFromID( _
                 ByVal ControlID As Long, , , , , ,) As String
    Select Case ControlID
        Case 1
            ISmartDocument_ControlCaptionFromID = _
                            "Inserire la ragione sociale ..."
        Case 101
            ISmartDocument_ControlCaptionFromID = _
                                    "Inserire l'indirizzo:"
        Case 201
            ISmartDocument_ControlCaptionFromID = _
                                    "Selezionare la città:"
        Case 301
            ISmartDocument_ControlCaptionFromID = _
                            "Selezionare il tipo di cliente:"
        Case 401, 402
            ISmartDocument_ControlCaptionFromID = _
                                    "Selezionare il logo:"
        Case Else
    End Select
End Property
Private Property Get ISmartDocument ControlTypeFromID
        (ByVal ControlID As Long, , ) As SmartTagLib.C_TYPE
    Select Case ControlID
        Case 1
            ISmartDocument_ControlTypeFromID =
                                        C_TYPE_TEXTBOX
         Case 101
            ISmartDocument_ControlTypeFromID =
                                        C_TYPE_TEXTBOX
         Case 201
            ISmartDocument_ControlTypeFromID =
                                          C_TYPE_COMBO
         Case 301
            ISmartDocument_ControlTypeFromID =
                                        C_TYPE_LISTBOX
         Case 401, 402
            ISmartDocument_ControlTypeFromID =
                                          C_TYPE_IMAGE
        Case Else
    End Select
End Property
```

Per inserire dei valori nei controlli ListBox, *Immagine* e *ControlBox* usiamo le seguenti.

```
Private Sub ISmartDocument_PopulateImage(ByVal
         ControlID As Long, , , , , , , ImageSrc As String)
    Select Case ControlID
        Case 401
            ImageSrc = strPath & "logo1.bmp"
        Case 402
            ImageSrc = strPath & "logo2.bmp"
    End Select
End Sub
Private Sub ISmartDocument_
                       PopulateListOrComboContent(ByVal _
  ControlID As Long, , , , , , , List() As String, Count As
                          Long, InitialSelected As Long)
   Select Case ControlID
    Case 201
        Count = 5
        ReDim List(1 To 5) As String
        List(1) = "Bari"
        ...
        List(5) = "Napoli"
        InitialSelected = -1
      Case 301
        Count = 5
        ReDim List(1 To 5) As String
        List(1) = "farmacista"
        ...
        List(5) = "meteorologo"
        InitialSelected = -1
    End Select
End Sub
```

A questo punto bisogna aggiungere il codice che imposta le azioni dei controlli e compilare la DLL.

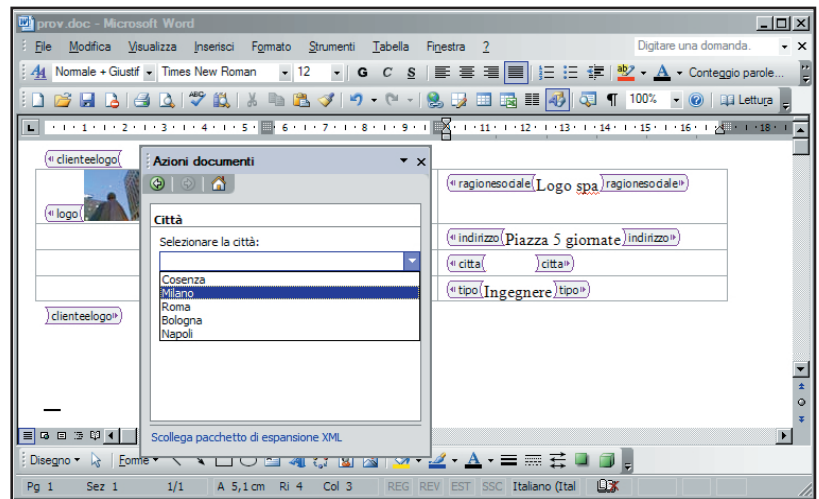



Fig. 4: **Il riquadro Azioni Documenti che mostra un combobox.**

CONCLUSIONI

Nello spazio a nostra disposizione non abbiamo potuto completare l'esempio, infatti, ancora dobbiamo spiegare come associare delle azioni ai controlli, come implementare il file *Manifest.xml* e come utilizzare il file che disabilita il controllo sulla certificazione del codice e perché è necessario.
Nel successivo appuntamento oltre a completare gli esempi ci occuperemo dell'*XML Expansion Packs* e della sicurezza.

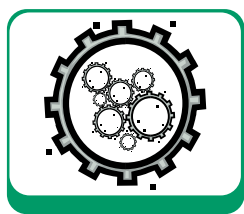
Massimo Autiero

MANIFEST

Per associare lo Smart Document alle applicazioni Office 2003 possiamo utilizzato un XML Espansion Packs composto dai file dello Smart Document è gestito dal file "Manifest.xml". Questo però non è l'unico modo per distribuire gli Smart Document, come vedremo nel prossimo appuntamento.

Programmiamo un agent per le nostre applicazioni

Un assistente a tua immagine

Nel numero scorso abbiamo programmato un assistente "prefabbricato". Oggi, ci cimenteremo nella costruzione e animazione di un personaggio da integrare nei nostri progetti!

Cominciamo col tranquillizzare chi si fosse perso l'appuntamento della volta scorsa. Per poter seguire questa puntata basterà procurarsi lo stretto necessario, disponibile gratuitamente all'indirizzo *http://www.microsoft.com/msagent/downloads/user.asp*. Per quanto riguarda le conoscenze "arretrate", basterà fare riferimento alla guida dell'SDK di Agent, o a uno dei tutorial disponibili in rete, per mettersi rapidamente al passo.
Ben diverso è il caso degli argomenti che toccheremo questa volta. Se la rete è prodiga di materiale sulla programmazione degli agenti tramite controllo o scripting, è quanto meno avara di informazioni circa la costruzione di un nuovo assistente.
La scarsa reperibilità di informazioni e tutorial può forse trovare giustificazione nella complessità intrinseca di un simile argomento: quando si mescolano arte, tecnica e animazione allo stesso tempo, stabilire un iter unico da seguire diventa impossibile.
Creare un personaggio può, infatti, richiedere numerosi e sofisticati programmi di grafica e animazione, oppure semplicemente buona volontà e Paintbrush.
Sta all'esperienza, al talento e all'inclinazione individuale la scelta dello strumento più opportuno.
Quest'articolo tratterà le fasi principali della realizzazione di un personaggio senza forzare l'uso di alcun programma specifico, anche se, doverosamente, vengono di volta in volta citati gli strumenti utilizzati.
In questa maniera l'autore spera di fare cosa gradita sia a chi ha gli strumenti a disposizione e vuole cimentarvisi in maniera professionale, sia a chi non sa usarli ma non vuol perdersi l'opportunità di seguire, comunque, l'avventura che questa puntata propone.

IL CHARACTER EDITOR

Qualunque strumento si utilizzi, l'indispensabile fucina per forgiare il nostro assistente è l'*ACE*: *Agent Character Editor,* disponibile, anch'esso gratuitamente, all'indirizzo *www.microsoft.com/msagent/downloads/developer.asp*.
Impareremo presto l'utilizzo di questo strumento, che, a dispetto del nome, non è un editor in senso stretto, quanto, piuttosto, un compilatore. Attraverso questo programma, non è infatti possibile leggere le informazioni contenute in un file *.ACS* (il file compilato di un assistente), come quelli di Merlin o Genie.
I file creati tramite ACE vengono salvati in formato *.ACD (Agent Character Description),* il quale, ad un analisi col blocco note, si rivela essere nient'altro che una serie di direttive circa locazioni dei file e delle animazioni necessari alla costruzione del personaggio. È per mezzo di un compilatore (interno al programma) che, successivamente, la sequenza di immagini e stati memorizzati in *.ACD* può essere convertita in un singolo file *.ACS*.

LA FASE CREATIVA

Siamo così arrivati alla prima fase del nostro progetto: l'atto creativo. Sembrerà banale, ma è fondamentale sapere in anticipo cosa vogliamo realizzare, e perché.
Il personaggio che abbiamo in mente dovrà soddisfare alcuni requisiti fondamentali: ri-

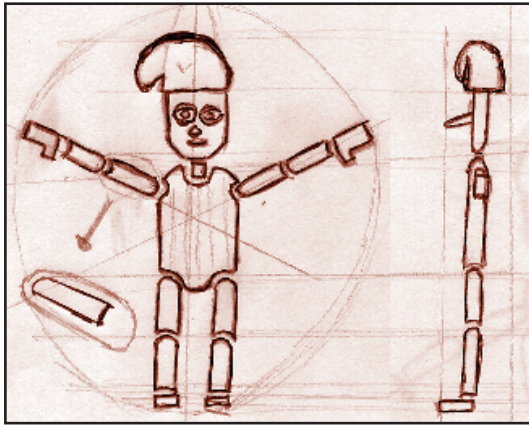



Fig. 1: *Alcuni bozzetti su carta dell'assistente Fili.*

sultare simpatico per l'utente, apparire fortemente caratterizzato, e prestarsi ad una ventina di animazioni diverse con grande facilità. É necessario, per questo motivo, visualizzare mentalmente il nostro personaggio che entra in scena, si muove, parla e saluta, si confonde e si annoia, e così via.
Pensando alla possibilità di interazione fra più personaggi, può, ad esempio, venire in mente una marionetta: questa dà l'opportunità, con piccole modifiche, di far nascere velocemente un piccolo teatrino di altri personaggi da 'mettere in scena'.
Più tempo si spende nella fase creativa, più è assicurata la caratterizzazione del personaggio: per la nostra marionetta pensiamo a grandi occhi espressivi (azzurri, in modo da risaltare meglio sul legno), un cappello verde (chino da un lato, in modo da non prendere troppo spazio), un naso appuntito e allungabile secondo la migliore tradizione 'collodiana'.
Diamogli un nome breve e significativo, ed ecco nascere l'assistente *"Fili"*. Abbiamo compiuto il primo passo importante, anche se quest'apparizione su carta ha, per il momento, ben poco di virtuale.

L'IMPLEMENTAZIONE TECNICA

A tutta questa progettazione tipica della fumettistica dobbiamo anche abbinare delle scelte tecniche: le vie per realizzare l'assistente, come ho già accennato, sono molte e richiedono l'uso di programmi e competenze differenti. É anche vero che ognuna di queste può adattarsi meglio a uno specifico tipo di personaggio, ragion per cui è bene spendere molto tempo nella scelta degli strumenti, considerando i programmi che si hanno a disposizione e confrontando i risultati di eventuali prove effettuate.
Prendiamo l'esempio di *Fili*: si tratta di un personaggio dai lineamenti volutamente squadrati, dai movimenti legnosi e dalle posizioni innaturali e quasi robotiche. Tutti questi fattori e la difficoltà di disegnare manualmente il legno che lo riveste in maniera coerente per tutti i fotogrammi delle animazioni, portano alla scelta di un programma di grafica tridimensionale.
Il programma scelto per *Fili* è 3D Max Studio: la realizzazione potrà così trarre vantaggio dalla struttura 'spezzata' degli arti del soggetto in questione. Non avendo gomiti, spalle, polsi e ginocchia, non ci sarà bisogno di prevedere complicate deformazioni nelle giunture, di "pesare" i vertici, di applicare un modificatore *skin* su una struttura di *bones*, o *phisique* sul *biped* di *Character Studio*.
Per ottenere una cinematica adatta ai nostri scopi basterà, invece, realizzare una struttura gerarchica di unioni ben studiata, come quella mostrata in Fig. 2.

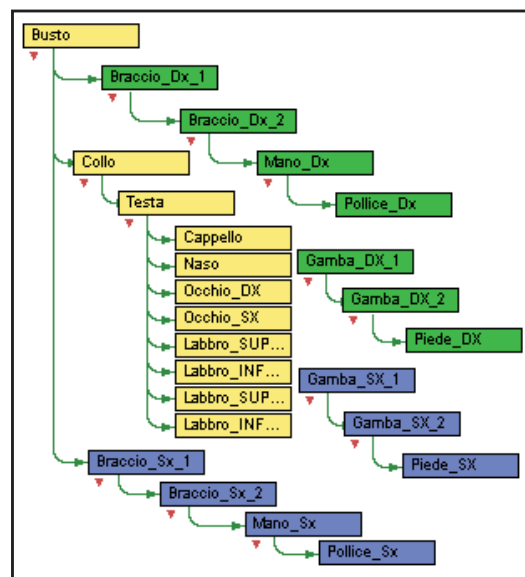
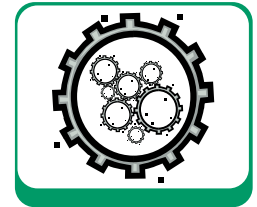



Fig. 2: *La struttura gerarchica delle componenti di Fili.*

Dallo stesso diagramma è anche possibile ricavare i nomi delle parti del corpo di *Fili*. Dal punto di vista della realizzazione grafica, si tratta principalmente di poligoni bidimensionali estrusi. Con questa semplice tecnica è possibile creare ogni componente ad eccezione degli occhi (due sfere), del naso (un cono "levigato"), e del cappello (ricavato da una sezione di cono cavo, modificata con una mappa di noise e inclinata dinamicamente tramite il modificatore *bend*).
Molta attenzione va anche prestata al tipo e

L'ALLEGATO

Nel file allegato su CD e sul Web è disponibile un piccolo ma significativo "laboratorio" per seguire l'articolo: sono presenti i frame che compongono le animazioni trattate e il file *Fili.acd* che le riunisce, ed è pienamente modificabile mediante *Character Editor.*

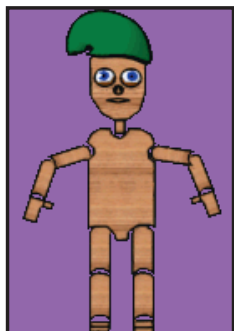
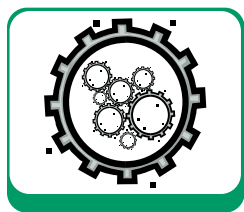



Fig. 3: Il rendering della restpose di Fili.

alle opzioni del rendering. Nel caso di un personaggio di fantasia (come il nostro) tendere al fotorealismo è sconsigliabile: meglio optare per una riproduzione del soggetto in stile fumetto.
In 3D Max Studio è disponibile, a tal proposito, il materiale *ink 'n paint,* che implementa una tipologia di rendering in cartoon-shading, con possibilità di definire il tipo di contorni (qualità e spessore dell'inchiostro), ed eventuali mappe di diffusione e *bump*.
A questo punto si tratta di mettere a posto gli ultimi elementi della scena. Impostiamo un colore di sfondo forte e non utilizzato dalla palette del personaggio: un viola brillante (RGB 255,0,255). Questo sarà il nostro colore di trasparenza, e, di conseguenza, dovremo prevedere un accorgimento particolare nell'uso dell'antialiasing, che dovrà essere applicato solo all'interno dell'area di disegno (nel nostro caso, l'*Ink 'n paint* si prende cura automaticamente di disattivare il *motion blur* e l'*antialiasing* esterno). Se l'antialiasing venisse applicato all'intera figura, infatti, l'assistente apparirebbe circondato da un fastidioso alone violaceo, effetto della contaminazione fra i contorni neri e il viola di trasparenza.
Facciamo assumere a *Fili* una posa adeguata e servizievole, e procediamo col rendering: ecco la *rest pose* del nostro assistente! Potete trovare quest'immagine nel codice di esempio allegato al CD *[file RestPose_16M.bmp]*.

Tavolozza
- Taglio medio ottimizzato
- Octree ottimizzata
- Standard/sicura per il Web

Metodo di riduzione
- Colore più simile
- Retinatura ordinata
- Diffusione di errore

Opzioni
- Incrementa i colori contrassegnati di: 1 (da 1 a 10)
- Includi colori di Windows
- Riduci scolorimento

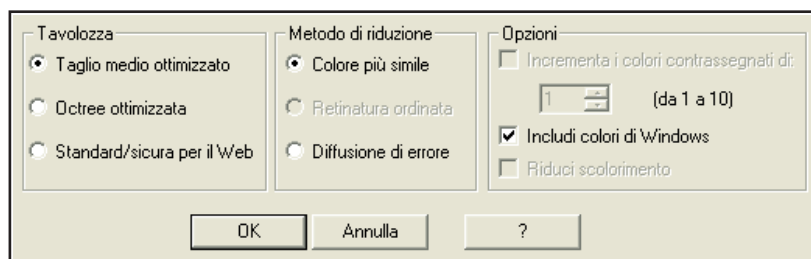
OK | Annulla | ?

Fig. 4: Le opzioni di riduzione della palette in Paint Shop Pro.

L'INTEGRAZIONE CON ACE

Qualunque sia la strada tecnica che abbiamo deciso di intraprendere, ora ci troviamo con un'immagine dell'assistente 'in posa' su sfondo viola. Per riuscire ad integrare il nostro frame con ACE, dobbiamo capire come questo 'interpreta' le animazioni e cosa richiede dalle nostre immagini.
Tutto il sistema degli Agent si basa sul concetto di 'animazione': ognuna di queste è realizzata attraverso una serie di fotogrammi (frames), ed è associata a uno o più stati *(states)*. Degli ultimi parleremo più avanti, mentre ora dobbiamo spendere un po' di tempo su frames e animazioni.
Il problema principale di integrazione che questi ci pongono è che tutti i frame di tutte le animazioni devono condividere necessariamente la stessa palette di base: le bitmap che produciamo, quindi, non solo avranno la limitazione di 256 colori a disposizione, ma dovranno inoltre condividere la stessa palette, per di più lasciando libere le prime e le ultime dieci posizioni ai colori standard di Windows. Avremo a disposizione, solo 236 colori effettivi, i quali saranno estremamente preziosi e dovranno essere valutati con molta cura.
Se questo problema non appare tanto minaccioso a chi si approccia 'manualmente' al disegno dell'assistente (236 colori, alla fin fine, rappresentano una risorsa vasta quando si possono scegliere volta per volta), per il nostro *Fili* è un bel grattacapo, dal momento che 3D Max Studio è sì in grado di generare una palette ottimizzata a 256 colori, ma non di riutilizzare la stessa per ogni immagine.
É quindi necessario fare ricorso ad un ulteriore programma, che riesca a gestire il tutto. Fra le scelte più comuni per questo tipo di esigenze figura il Paint Shop Pro, che può vantare dalla sua un ottimo sistema di gestione delle palette (nonché una buona versione trial, per chi non disponesse del prodotto).
Grazie a quest'applicazione, potremo effettuare aggiunte post-rendering quando sarà necessario, e gestire i frames con l'animation shop, ad esempio per ottimizzare al massimo le dimensioni dell'assistente, ritagliando opportunamente l'area animata, in modo automatico.
Per quanto riguarda la nostra restpose (ottenuta in 3D Max Studio a 16 milioni di colori) possiamo aprirla con Paint Shop Pro e convertirla a 256 colori *(Shift+Ctrl+3),* scegliendo le opzioni indicate in Fig. 4.
Verrà automaticamente generata la palette ottimizzata, che provvederemo a salvare (menù *Colori/Salva tavolozza*) in un file *.pal*. Da ora in poi, per ogni frame generato da 3D Max Studio, basterà caricare quest'ultimo, per ottenere la conversione ottimizzata del disegno. Avremo così ciò che ci eravamo prefissi: un'u-

OFFICE

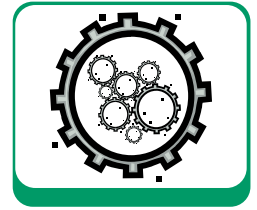
Gli agenti costruiti con ACE possono essere utilizzati anche per Microsoft Office (solo per la versione 2000 e successive). A tal fine occorre supportare le animazioni standard di Office (*File/New/Office Assistant*), usare l'apposita palette
www.microsoft.com/msagent/download/assistpalet.bmp
e immettere il file nella cartella *C:\...\Application Data\Office\Actors.*

nica palette per tutti i disegni (in *RestPose-.Bmp* è presente l'immagine convertita e ritagliata).

LA PRIMA ANIMAZIONE

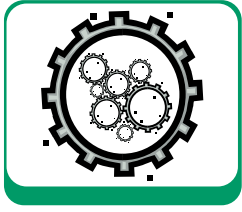
É quindi con lo spirito sollevato, proprio di chi ha risolto gran parte dei suoi problemi, che ci accingiamo alla realizzazione della prima animazione del nostro assistente!
Aprendo l'ACE e cliccando sul menù *File/New /Default Character,* otteniamo uno scheletro di tutte le animazioni e gli stati, così come vengono associati secondo il modo 'standard' utilizzato da Windows.
La pagina iniziale, inoltre, presenta dei campi da compilare (nome assistente, descrizione, extra, etc…): questi dati sono importanti, poiché appaiono nella finestra di scelta dell'assistente, quando viene richiamato il metodo *ShowDefaultCharacterProperties* del controllo Agent.
Altrettanto importante è la schermata associata all'elemento *animation*, all'interno della quale dobbiamo indicare il colore (viola) di trasparenza.
Espandendo il ramo *Animations* è possibile rintracciare l'animazione *RestPose*, che contiene un unico fotogramma e viene richiamata dal server ogni volta che l'assistente è in posizione neutra. È a quest'animazione che dovremo associare il nostro disegno appena creato.
Clicchiamo col tasto destro su *RestPose* e scegliamo *'New Frames from Images'* dal menù di pop-up, indi apriamo il file *RestPose.Bmp*.
L'immagine del nostro simpatico assistente apparirà sulla preview di destra, anche se deformata. Ciò non deve spaventare più di tanto: accade perché l'assistente ha dimensioni diverse da quelle standard (128x128), ma in fase di esecuzione l'assistente sarà proporzionato normalmente (anche se apparirà più grande di Genie e Merlin).
Per vedere se tutto è andato come deve, apriamo il menù *File/Build Character* per compilare l'assistente, che salveremo come *Fili.acs* in *C:\Windows\ MsAgent\Chars:* la cartella predefinita per gli assistenti professionali!
Per provare il nostro operato, possiamo usare l'Hello World costruito la volta scorsa, con l'unico cambiamento del nome dell'assistente: non più *"Merlin.acs",* ma *"Fili.acs".*
Se tutto è andato come deve, apparirà la nostra marionetta, esclamando *"Ciao, mondo!".* In realtà, il verbo 'esclamare' è esagerato, dal momento che *Fili* non emetterà alcun suono, né, tantomeno, muoverà le labbra in alcun modo.

IL PARLATO E GLI OVERLAY

Non è certamente colpa di *Fili*, se sta fermo e muto. Siamo noi a dovergli dare la favella, cliccando sull'elemento *animation* e quindi spuntando la casella *"Use syntesized speech for voice output".* Apparirà dinamicamente la nuova linguetta *"Voices",* all'interno della quale potremo stabilire il motore da utilizzare, il pitch e la velocità del parlato.
Dal momento che *Fili* ha le sembianze di un bambino, proporrei la combinazione: *"Adult Female #1 Italian (L&H)"* (scaricato nella scorsa puntata), con 166 Hz e 145 WPM.
Basta compilare l'assistente ed eseguire il programma di prova, per accorgersi che, *Fili* è pronto a pronunciare concretamente tutto ciò che dice.
Rimane, però, il problema del *lip-sync,* ovvero della sincronizzazione delle labbra al parlato: questo è un elemento importante, perché aggiunge molti punti alla credibilità del nostro assistente.
Chiunque abbia provato a realizzare un buon motore di *lip-sync* conosce bene le problematiche che pone una sfida del genere: fortunatamente Agent ci facilita il compito, assegnando automaticamente ad ogni fonema di base una fra le sette posizioni disponibili per la bocca.
Cliccando sul *Frame1* di *RestPose* è infatti possibile notare una nuova linguetta *(Overlays),* che all'interno contiene i sette slot per le posizioni suddette. Basta disegnare per ogni stato un'opportuna posizione della bocca, in accordo con l'esempio mostrato dall'applicazione in basso a destra. Per *Fili* tali disegni sono contenuti nei file il cui nome comincia con *"bocca".* Si può notare come queste immagini siano solo dei piccoli rettangoli: l'idea è quella di sovrapporli al disegno di base, coprendo l'area della bocca. Ciò permette di risparmiare molto spazio e diversi rendering; immagini di questo tipo prendono il nome di overlay. Quando si immette un overlay è necessario specificare anche un offset, ovvero la posizione delle coordinate *x* e *y* dell'angolo superiore sinistro in cui si vuole sovrapporre il disegno: per gli overlay della bocca di *Fili* tali valori sono *x=52* e *y=55.*Possiamo ora compilare l'assistente e goderci un parlato preciso e professionale!

NOTA

Ad un singolo stato possono essere assegnate quante animazioni si desidera. In un simile caso il server sceglierà automaticamente, in modo casuale, una delle animazioni possibili di volta in volta. Sfruttando questo principio in congiunzione con il *branching* è possibile creare assistenti dalle reazioni sorprendentemente varie.

NOTA SUL CODICE

Quando un personaggio (come il nostro) non supporta ancora l'intero animation set, è un'ottima pratica richiamare la restpose prima di qualunque azione. Il server potrebbe, infatti, richiamare stati non associati ad animazioni, con conseguente "bloccaggio" dell'assistente (ad esempio, questi potrebbe parlare senza muovere la bocca).

LA CONDIZIONE DELL'AGENT

Come promesso, eccoci a parlare degli stati: cosa sono? E a cosa servono? Per rispondere a queste e ad altre domande, occorre conoscere come il server di Agent fa girare le istanze degli assistenti: a seconda di quello che viene richiesto dal programmatore attraverso funzioni esplicite (del tipo *.Show*) e di ciò che accade implicitamente (ad esempio l'inattività prolungata), il server assegna agli agenti degli stati: i 16 elencati dal *Character Editor.*

GLOSSARIO

RETURN ANIMATION

Quando un'animazione non finisce in posa neutrale è bene definire anche una *return animation*, usando l'apposito campo disponibile per ogni 'elemento animazione' nella lista. Questa viene richiamata dal server, dopo il termine della stessa, e deve riportare il personaggio, gradualmente, alla restpose; ciò è volto ad evitare un brusco e innaturale 'salto' di ritorno.

EXIT BRANCHING

La *return animation* non fornisce sufficienti garanzie di funzionamento quando, usando il *branching*, si crea un'animazione a ciclo infinito. In tal caso è possibile usare l'apposito *Exit Branching*, che prevede un'uscita specifica per ogni frame, consentendo transizioni più precise. Un esempio di tale tecnica è fornito nell'animazione searching di *"Fili.acd"*.

L'AGENTRY

Uno dei siti di riferimento per i creatori di ACS è sicuramente http://agentry.net, che fornisce applicativi che sfruttano questa tecnologia, un'impressionante collezione di oltre 400 assistenti (gratis o a pagamento), e può diventare un'ottima vetrina per le nostre migliori creazioni.

Ad ognuno di questi stati è possibile assegnare una o più animazioni (per quest'ultime, invece, non c'è alcun limite: è possibile creare quante animazioni si vuole e assegnarle a tutti gli stati che si desiderano coprire). Quando, ad esempio, l'assistente non viene usato per lungo tempo, viene generato uno stato di *Idle*, dapprima lieve *(IdlingLevel1)*, poi più 'grave' *(IdlingLevel2)*, fino alla notifica della sonnolenza completa *(IdlingLevel3)*.
Per standard, questi stati di *Idle* sono coperti da alcune animazioni: *"Blink"*, ad esempio, viene richiamata sia per lo stato di *Idle1*, che per quello di *Idle2*. Possiamo verificare che, implementando l'animazione di *blink*, l'assistente sbatte automaticamente gli occhi dopo un periodo di inattività. Per realizzare l'animazione, basta inserire su tre frame consecutivi l'immagine di restpose. Sul secondo, piazziamo un overlay degli occhi chiusi *(File Occhi_overlay.BMP)* al posto giusto, e il gioco è fatto (per aggiungere un overlay al frame basta premere il pulsante *Add Image* sulla toolbar contenuta nel frame *Images*). É anche consigliato aumentare il numero di centesimi di secondo del frame, impostando a 20 il campo *Duration*.

SUL FORUM

Per qualsiasi suggerimento, critica, approfondimento, l'autore può essere contattato sul Forum del sito di ioProgrammo.

IL BRANCHING

Quando, in Windows XP, facciamo cercare un file a Merlin, costui tira fuori una sfera di vetro e inizia la sua divinazione. Questa viene portata avanti, finché il file non viene effettivamente trovato dal programma. Il diabolico stratagemma non ci inganna, dato che abbiamo visto nella scorsa puntata che questo genere di trucchi può essere realizzato richiamando un'animazione continua, e passando il metodo *Stop* quando necessario. Ma com'è possibile realizzare un'animazione continua? Questa domanda è non solo legittima, ma necessaria, dal momento che alcune animazioni (come la *"searching"*, per esempio) sono progettate per essere dichiaratamente continue.
Tutto questo è possibile grazie al branching: per ogni frame, si possono impostare fino a tre 'salti' ad un altro frame (purtroppo, soltanto nei limiti dell'animazione corrente), a seconda di una probabilità che può andare da 0% (salto non presente) a 100% (che equivale a un GoTo). Per realizzare un'animazione continua è quindi sufficiente impostare nell'ultimo frame un branching 100% che riconduca alla prima. In realtà il sistema fornito da Agent permette loop molto più raffinati, in maniera che la natura meccanica della ripetizione non risulti troppo evidente: impostando dei salti con probabilità intermedie, è possibile far prendere all'animazione strade diverse e rompere la monotonia del ciclo. Si può studiare un simile comportamento consultando l'animazione *"Searching"* del file *Fili.asd*, fornito nel codice di esempio.

CONCLUSIONI

Si conclude qui il nostro viaggio nel mondo degli agenti, ormai non più… segreti. Tramite questo giro panoramico, forse qualche lettore potrà trovare degli spunti interessanti, magari condividendo le proprie esperienze sul forum di ioProgrammo. E quantomeno, potrà sicuramente osservare con occhi più consapevoli la "dannata graffetta di Office!".

Roberto Allegra

Impariamo a navigare lo spazio 3D con DirectX e C#

Trasformazioni in .NET

Dopo aver visto il disegno di semplici primitive su schermo attraverso i Vertex Buffer e gli oggetti GraphicsStream, in questo numero realizziamo il movimento degli oggetti nello spazio 3D.

Dopo aver inviato tutti i vertici del nostro disegno a DirectX, avvengono delle elaborazioni interne per trasportare le coordinate spaziali da noi inserite (3D) in coordinate dello schermo (2D), come mostrato nel disegno in Fig. 1: processo, ripetiamolo, chiamato renderizzazione.

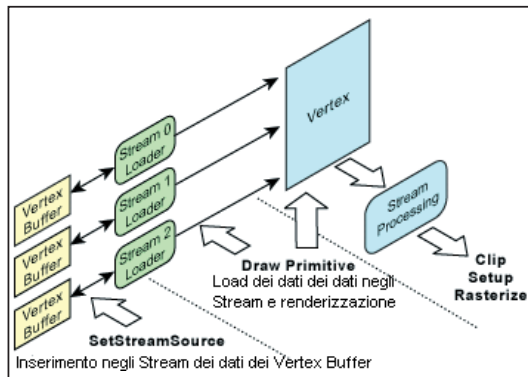



Fig. 1: Processo di Render.

Quando DirectX termina i calcoli per ogni pixel, invierà questi dati dalla memoria allo schermo: questo processo è chiamato *rasterizzazione*. In definitiva abbiamo due fasi del processo; la prima si occupa della *geometria* dell'oggetto che vogliamo rappresentare (vertici, triangoli, ecc); la seconda del trasporto sulla memoria dello schermo dei singoli pixel che devono essere rappresentati. La prima parte, essendo di natura geometrica, è governata dal capitolo della geometria analitica che si occupa delle *trasformazioni*. Cerchiamo di capire assieme di cosa si tratta.

TRASFORMAZIONI

Una trasformazione è un formalismo che ci permette di rappresentare con operazioni matematiche il processo che determina la posizione di un punto. Questo processo può essere una rotazione, una traslazione, una proiezione, ecc; le trasformazioni fondamentalmente alterano la disposizione di punti all'interno di uno spazio. Fra tutti i tipi di trasformazioni si distinguono:

Le **trasformazioni proiettive,** le quali conservano gli allineamenti fra i punti e le figure. Vengono usate nella trasposizione in 2D dello spazio 3D attraverso le regole della prospettiva.
Le **trasformazioni affini**, che oltre ad esser proiettive e quindi a conservare gli allineamenti, conservano anche i rapporti fra le distanze degli oggetti.
Le **trasformazioni metriche** infine, oltre ad essere affini, conservano anche le distanze esatte fra gli oggetti.

Per chiarire meglio, consideriamo il nostro triangolo (base di ogni rappresentazione grafica): quando ne facciamo una rappresentazione prospettica il nostro triangolo può cambiare forma ma i suoi lati saranno sempre dei segmenti di retta: la trasformazione è proiettiva. Quando ne facciamo una rappresentazione in scala non solo i lati rimangono segmenti di retta, ma ne conserviamo anche i rapporti tra distanze: è una trasformazione affine. Quando le facciamo una fotocopia conserviamo anche ogni singola misura: le lunghezze dei lati, l'area, ecc.: è una trasformazione metrica. Tutte queste trasformazioni come insegna un branca della geometria analitica, si possono rappresentare quali operazioni fra i vettori che rappresentano le coordinate dei punti e le matrici che rappresentano le caratteristiche della trasformazione. Il formalismo generale è

$$P' = P * M$$

Questo formalismo ancora non ci dice in che tipo di

Impegno

Tempo di realizzazione

spazio (2D, 3D) stiamo lavorando e quindi si perfeziona nel seguente modo:

*3D: P'(1,3) = P(1,3) * M(3,3) + T(1,3)*
*2D: P'(1,2) = P(1,2) * M(2,2) + T(1,2)*

Come si vede con le coordinate spaziali che conosciamo occorrerà applicare due operazioni matematiche: una moltiplicazione di matrici ed una somma di matrici. Una ottimizzazione di questo processo si ha con l'introduzione delle coordinate omogenee che ci permetteranno di ridurlo ad una sola operazione:

*3D: P'(1,4) = P(1,4) * M(4,4)*
*2D: P'(1,3) = P(1,3) * M(3,3)*

Scendiamo ora più nel dettaglio e cerchiamo di capire come lavorare con le matrici e *soprattutto come usarle nei nostri programmi di grafica 3D.*

MATEMATICA DELLE MATRICI

Una matrice può essere pensata come una scatola di equazioni lineari, ma per capire bene questo concetto, dobbiamo prima capire come le matrici possono essere moltiplicate fra loro e imparare le regole di queste moltiplicazioni. Per farla breve, due matrici possono essere moltiplicate fra loro solo, e solo se, la prima ha lo stesso numero di colonne del numero di righe della seconda:

*A[5][3] * B[3][7], A[1][4] * B[4][4], A[3][2] * B [2][9]*

sono tutte moltiplicazioni valide, mentre

*A[3][5] * B[3][7], A[4][1] * B[4][4], A[2][3] * B [2][9]*

non sono moltiplicazioni possibili. Per moltiplicare due matrici dobbiamo prima considerare ogni riga della matrice A come un vettore, in questo caso di tre elementi: *aRiga1(a11, a12, a13); aRiga2(a21, a22, a33); aRiga3(a31, a32, a33)* e ogni colonna della matrice *B* come un vettore *bColonna1(b11,b21,b31); bColonna2(b21,b22,b23); bColonna3(b31,b32,b33).* A questo punto moltiplichiamo i vettori nel seguente modo: il primo vettore della matrice *A (aRiga1)* con il primo vettore della matrice *B (bCol1)*, il primo vettore della matrice *A (aRiga1)* con il secondo vettore della matrice *B (bCol2)*, il primo vettore della matrice *A (aRiga1)* con il terzo vettore della matrice B (bCol3); e così via per ogni riga (o vettore) di A. Lo schema in Fig. 2 indica esattamente come eseguire la moltiplicazione matriciale.
Quel puntino che vedete fra ogni vettore dentro la matrice risultato, è una speciale operazione matematica detta moltiplicazione vettoriale, e significa che bisogna eseguire questo tipo di operazione:

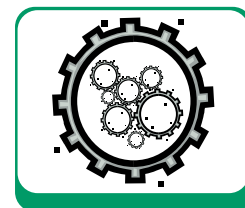
$$\begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{bmatrix} \times \begin{bmatrix} b11 & b12 & b13 \\ b21 & b22 & b23 \\ b31 & b32 & b33 \end{bmatrix} = \begin{bmatrix} aRiga1 \bullet bCol1 & aRiga1 \bullet bCol2 & aRiga1 \bullet bCol3 \\ aRiga2 \bullet bCol1 & aRiga2 \bullet bCol2 & aRiga2 \bullet bCol3 \\ aRiga3 \bullet bCol1 & aRiga3 \bullet bCol2 & aRiga3 \bullet bCol3 \end{bmatrix}$$

Fig. 2: **Moltiplicazione matriciale.**

*aRiga1 * bCol1 = a11 * b11 + a12 * b21 + a13 * b31*

Iniziamo finalmente a tradurre in codice C# quanto detto fino ad ora, scrivendo una funzione che effettui una moltiplicazione fra due matrici 4x4:

```
R(4,4) = A(4,4) * B(4,4)
public Matrix MoltiplicaMatrici(Matrix a, Matrix b)
{ Matrix r;
  r.M11 = a.M11*b.M11 + a.M12*b.M21 +
                        a.M13*b.M31 + a.M14*b.M41;
  r.M12 = a.M11*b.M12 + a.M12*b.M22 +
                        a.M13*b.M32 + a.M14*b.M42;
  r.M13 = a.M11*b.M13 + a.M12*b.M23 +
                        a.M13*b.M33 + a.M14*b.M43;
  r.M14 = a.M11*b.M14 + a.M12*b.M24 +
                        a.M13*b.M34 + a.M14*b.M44;
  r.M21 = a.M21*b.M11 + a.M22*b.M21 +
                        a.M23*b.M31 + a.M24*b.M41;
  r.M22 = a.M21*b.M12 + a.M22*b.M22 +
                        a.M23*b.M32 + a.M24*b.M42;
  r.M23 = a.M21*b.M13 + a.M22*b.M23 +
                        a.M23*b.M33 + a.M24*b.M43;
  r.M24 = a.M21*b.M14 + a.M22*b.M24 +
                        a.M23*b.M34 + a.M24*b.M44;
  r.M31 = a.M31*b.M11 + a.M32*b.M21 +
                        a.M33*b.M31 + a.M34*b.M41;
  r.M32 = a.M31*b.M12 + a.M32*b.M22 +
                        a.M33*b.M32 + a.M34*b.M42;
  r.M33 = a.M31*b.M13 + a.M32*b.M23 +
                        a.M33*b.M33 + a.M34*b.M43;
  r.M34 = a.M31*b.M14 + a.M32*b.M24 +
                        a.M33*b.M34 + a.M34*b.M44;
  r.M41 = a.M41*b.M11 + a.M42*b.M21 +
                        a.M43*b.M31 + a.M44*b.M41;
  r.M42 = a.M41*b.M12 + a.M42*b.M22 +
                        a.M43*b.M32 + a.M44*b.M42;
  r.M43 = a.M41*b.M13 + a.M42*b.M23 +
                        a.M43*b.M33 + a.M44*b.M43;
  r.M44 = a.M41*b.M14 + a.M42*b.M24 +
                        a.M43*b.M34 + a.M44*b.M44;
return r;}
```

In realtà non avremo quasi mai bisogno di scrivere le nostre funzioni di moltiplicazione fra matrici in C#, perché grazie alla libreria D3DX, DirectX espone moltissime utili funzioni per operare su matrici (ad esempio *Matrix.Multiply()* per moltiplicare due matrici; *Matrix.RotateX()* per rotazioni lungo l'asse *X*; *Matrix.Scale()* per le proporzioni, ecc). Ora scopria-

MATRICI IDENTITÀ

Una matrice particolare è quella di identità che è un po' come il numero 1 nella moltiplicazione, ossia ha la particolarità che, quando moltiplicata per un vettore, restituisce come risultato lo stesso vettore per la quale era stata moltiplicata, senza alterarlo in alcun modo. Una matrice identità è composta da tutti zeri, tranne una riga diagonale di uno, come mostra la figura sotto. È bene, quando usiamo le matrici di DirectX in oggetti più complessi, inizializzarle sempre con matrici identità, e quindi neutre (il luogo ideale per effettuare ciò è dentro il costruttore di una classe).

Matrice identità 3x3.

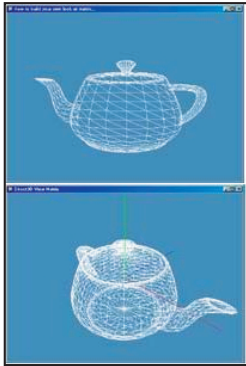
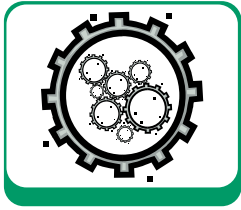



Fig. 3: Semplice esempio di trasformazione.

I QUATERNIONI

Oltre all'uso delle matrici, esiste anche un altro modo in DirectX per muovere gli oggetti nello spazio 3D: i quaternioni. Sir William Rowan Hamilton inventò i quaternioni (un'estensione dei numeri complessi) nel 1843, ma solo nel 1985 questi furono introdotti da Ken Shoemake nel campo della computer grafica. Per approfondimenti su questo argomento vi consiglio di visitare www.siggraph.org

mo finalmente in che modo tutta questa matematica possa aiutare i programmatori 3D per effettuare delle trasformazioni come ad esempio quella rappresentata in Fig. 3. Le matrici ci permettono di immagazzinare una serie di equazioni, in modo che quando un vettore è moltiplicato per una matrice, otteniamo come risultato un vettore "trasformato". Questo è molto importante perché avremo bisogno di applicare ogni sorta di trasformazione (rotazioni, traslazioni, ecc) alle nostre coordinate 3D. Quindi tutto ciò di cui abbiamo bisogno è individuare la matrice di queste trasformazioni e moltiplicarla poi per i nostri vettori o coordinate 3D ottenendo i nuovi vettori e le coordinate. Senza scendere nel dettaglio matematico e dando quindi sempre per scontato che si conoscano le equazioni delle geometria analitica, le matrici per lavorare con coordinate omogenee nel nostro spazio 3D sono quelle rappresentate in Fig. 4.

Fig. 4: Matrici omogenee 4x4 di trasformazione.

Concludiamo infine sottolineando come queste matrici 4x4 contengono nella parte superiore sinistra 3x3 una sottomatrice per le informazioni sulle rotazioni e proporzioni/scaling, nei primi tre valori dell'ultima riga le informazioni per le traslazioni, e nell'ultima colonna i valori 0,0,0,1 (questo perché vogliamo che il quarto elemento dei nostri vettori 4D omogenei resti invariato durante le moltiplicazioni: deve sempre essere uguale ad uno). Traducendo in codice quanto detto fino ad ora, potremmo a questo punto iniziare a sviluppare una classe (che amplieremo nei prossimi articoli) nella quale incapsulare le funzioni che possono esserci utili durante lo sviluppo di applicazioni 3D con DirectX:

```
public class MyUtility
{ #region Funzioni Matriciali
  public bool MuoviOggetto(float scalaX, float scalaY,
    float scalaZ, int angleX, int angleY, int angleZ, float
               posX, float posY, float posZ, Device dev)
  { //Variabili di appoggio
    float CosRx;
    float CosRy;
    float CosRz;
    float SinRx;
    float SinRy;
    float SinRz;
    //Fine variabili di appoggio
   float rad = 3.14/180;
    //Valore di conversione gradi/radianti
    //La matrice che conterrà tutte le informazioni
                       sulla trasformazione da eseguire
   Matrix TempMat;
   CosRx = Math.Cos(angleX * rad);
   CosRy = Math.Cos(angleY * rad);
   CosRz = Math.Cos(angleZ * rad);
   SinRx = Math.Sin(angleX * rad);
   SinRy = Math.Sin(angleY * rad);
   SinRz = Math.Sin(angleZ * rad);
   TempMat.M11 = (scalaX * CosRy * CosRz);
   TempMat.M12 = (scalaX * CosRy * SinRz);
   TempMat.M13 = -(scalaX * SinRy);
   TempMat.M21 = -(scalaY * CosRx * SinRz) +
                     (scalaY * SinRx * SinRy * CosRz);
   TempMat.M22 = (scalaY * CosRx * CosRz) +
                     (scalaY * SinRx * SinRy * SinRz);
   TempMat.M23 = (scalaY * SinRx * CosRy);
   TempMat.M31 = (scalaZ * SinRx * SinRz) +
                     (scalaZ * CosRx * SinRy * CosRz);
   TempMat.M32 = -(scalaZ * SinRx * CosRz) +
                     (scalaZ * CosRx * SinRy * SinRz);
   TempMat.M33 = (scalaZ * CosRx * CosRy);
   TempMat.M41 = posX;
   TempMat.M42 = posY;
   TempMat.M43 = posZ;
   TempMat.M44 = 1;
    Applico la matrice risultante al device di disegno
   dev.Transform.World = TempMat; // }
#endregion }
```

TRANSFORMATION PIPELINE

Ora che abbiamo compreso come avvengono le trasformazioni, studiamo cosa accade in Direct3D. Il primo passaggio, come mostra la Fig. 5 è quello dell'inserimento (nel vertex buffer) dei nostri vertici attraverso le loro coordinate iniziali. Dopo di che avviene un processo di trasformazioni che si concluderà con la rasterizzazione su schermo del nostro spazio. Quello che abbiamo fatto nell'articolo precedente è stato proprio questo, ignorando per intero il processo di trasformazioni (chiamato simpaticamente da Microsoft *Transformation Pipeline*, ossia il tubo delle trasformazioni). In questo processo si possono però applicare 3 matrici 4x4 che sono rispettivamente la *World Matrix*, la *View Matrix* e la *Projection Matrix*. La *World Matrix* indica le trasformazioni che avvengono nel nostro mondo (ad esempio la rotazione vera e propria dei nostri oggetti 3D all'interno delle coordinate spaziali). La *View Matrix* specifica le trasformazioni dovute dallo spo-

stamento del nostro punto di osservazione dello spazio (ossia specifica la posizione e l'orientamento della telecamera virtuale che si muove nello spazio e attraverso la quale noi vediamo le scene rappresentate sullo schermo). La *Projection Matrix* serve infine per passare dalle coordinate 3D virtuali (catturate dalla telecamera) alle vere coordinate 2D fisiche dello schermo. Come si vede dal disegno qui sopra, un ultimo passaggio è presente nella nostra *Transformation Pipeline,* ossia il *"Clipping* e *Viewport Scaling",* responsabile della selezione dei vertici che si vedranno sul nostro schermo (ad esempio i vertici posti davanti alla telecamera) rispetto a quelli da scartare (vertici posti dietro la telecamera o fuori dal suo angolo di visuale): la geometria non visibile può essere scartata (anzi, deve esserlo per motivi di performance) perché non contribuirà alla rappresentazione finale sullo schermo.

EFFETTUARE TRASFORMAZIONI

Abbiamo appena visto come tutti i vertici in Direct3D passino attraverso la Transformation Pipeline e siano sottoposti quindi a tre diverse trasformazioni. Per inserire le tre matrici di trasformazione che verranno usate da Direct3D in questo processo si usano le proprietà della classe *Device.Transform:*

```
device.Transform.World = [prima matrice 4x4];
device.Transform.View = [seconda matrice 4x4];
device.Transform.Projection = [terza matrice 4x4];
```

Fin qui tutto dovrebbe essere chiaro. Prima di vedere quali specifiche matrici dovremo usare, è necessario chiarire il fatto che mentre la *World Matrix*, in una applicazione che usa dinamicamente la grafica 3D, sarà in continuo aggiornamento per rispecchiare i movimenti degli oggetti nello spazio, e la *View Matrix* sarà aggiornata per rispecchiare lo spostamento dell'osservatore nel mondo virtuale, la matrice di Projection non viene mai generalmente modificata, perchè i rapporti fra il mondo 3D e la sua rappresentazione sullo schermo 2D sono sempre gli stessi (a meno di non voler creare dei voluti effetti di distorsione e alterazione della prospettiva).

WORLD MATRIX

Una *World Matrix* non è altro che una matrice 4x4 che contiene le informazioni riguardanti lo scaling, le traslazioni e rotazioni, che abbiamo visto poco fa, da applicare al nostro mondo 3D. Come già detto, D3DX contiene molte funzioni utilissime che ci risparmieranno di scrivere codice per il popolamento manuale delle nostre matrici. Ad esempio la funzione *Matrix.RotationY* accetta come parametro l'angolo di rotazione e restituisce una matrice 4x4 che effettuerà questa trasformazione. Fra le utilissime funzioni di questa classe ricordiamo:

Matrix.RotationX
Matrix.RotationY
Matrix.RotationZ
Matrix.Scale
Matrix.Translate
Matrix.RotationYawPitchRoll
Matrix.AffineTransformation

Inoltre, D3DX si preoccupa anche di fornirci i metodi per lavorare con le matrici come ad esempio:

Matrix.Multiply
Matrix.Equals
Matrix.Invert

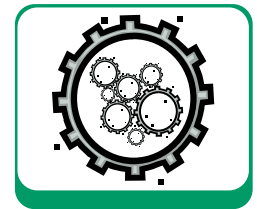
VIEW MATRIX

Creare la *View Matrix* consiste nel descrivere l'orientamento e la posizione corrente della nostra telecamera virtuale come un nuovo sistema di coordinate. Possiamo effettuare ciò specificando tre vettori che rappresenteranno gli assi *X, Y* e *Z* di questo sistema come mostrato in Fig. 5. Questi tre vettori prendono anche il nome di *Vettore Right (X), Vettore Up (Y)* e *Vettore Look (Z).* Ci sono numerosi modi per calcolare il valore di questi vettori, compreso l'uso di quaternioni e altri complessi sistemi matematici, ma per quanto ci concerne è sufficiente usare una funzione di D3DX che prende in input i tre vettori *Right, Up* e *Look* definiti come sempre attraverso una coordinata 3D:

Fig. 5: **Trasformation Pipeline.**

```
Matrix.LookAtLH(new Vector3(0.0f, 3.0f,-5.0f),
          new Vector3(0.0f, 0.0f, 0.0f),
          new Vector3(0.0f, 1.0f, 0.0f));
```

PROJECTION MATRIX

Come già detto questa matrice serve per trasformare (prospetticamente) la geometria 3D in geometria 2D per poterla disegnare sul nostro schermo in due dimensioni: se ad esempio su schermo vediamo gli oggetti distanti più piccoli di quelli vicini è grazie all'uso di questa matrice. Per costruire questa matrice è necessario specificare un angolo di vista *(FoV – Field Of View)*, un aspect ratio, un piano di clipping vicino e un piano di clipping lontano. La funzione

COSA C'È DIETRO BEGINSCENE() ED ENDSCENE()

***BeginScene()* fa si che il sistema (Direct3D) controlli le sue strutture di dati interne, la disponibilità e la validità delle superfici di rendering ed imposta infine un flag per segnalare che è in atto il disegno di una scena. *EndScene()* ripristina questo flag, effettua un flush dei dati di disegno immessi e verifica che le superfici di disegno siano corrette.**

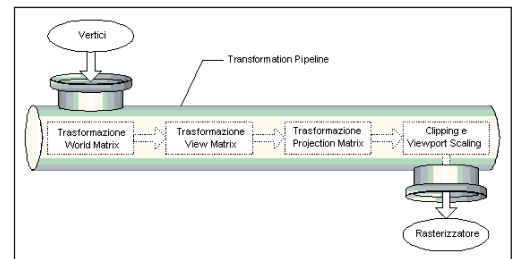
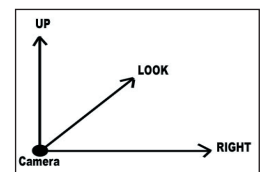
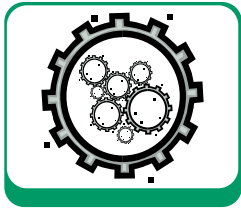



Fig. 6: **Vettori Up, Look e Right della View Matrix.**

D3DX che restituisce una matrice di questo tipo è:

```
Matrix.PerspectiveFovLH((float)Math.PI / 4, 1.0f, 1.0f,
                                                100.0f);
```

L'angolo di vista è importante per stabilire se la geometria che si trova non esattamente di fronte a noi rientrerà nella nostra visuale o meno.

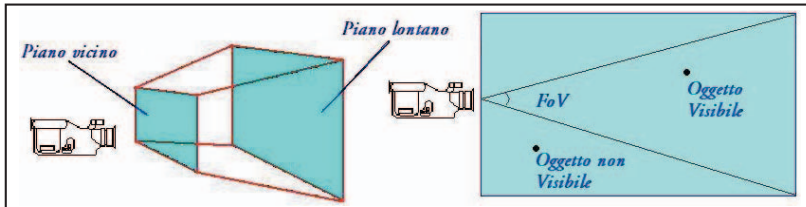



Fig. 7: ***Raffigurazione della Projection Matrix.***

Il parametro *Aspect Ratio* non è altro che l'altezza della superficie sulla quale disegniamo diviso per la sua larghezza. I piani di clipping servono per scartare la geometria che rispetto alla telecamera risulta essere troppo vicina (da problemi di rendering: se un punto si trova nella stessa coordinata Z di profondità avvengono delle divisioni per zero che mandano in errore l'intero sistema) o troppo lontana (inutile rappresentare qualcosa che non è visibile o sarebbe ridotto a meno di un pixel).

Fig. 8: ***Sequenza di immagini tratta dal nostro programma in esecuzione.***

IL NOSTRO ESEMPIO

Dopo tanta teoria, è finalmente arrivato il momento in cui vediamo ripagati i nostri sforzi. Nella funzione *Render()* del progetto del precedente articolo, subito dopo la chiamata a *device.BegineScene()* inseriamo una nuova funzione che useremo per applicare tutte le trasformazioni che vogliamo. Chiameremo la funzione *AggiornaMatrici e* sarà così semplicemente implementata:

```
private void AggiornaMatrici ()
{  // WORLD MATRIX: ruotiamo continuamente l'oggetto.
   device.Transform.World = Matrix.RotationAxis(
      new Vector3((float)Math.Cos(Environment.TickCount
       / 250.0f),1,(float)Math.Sin(Environment.TickCount
          / 250.0f)), Environment.TickCount / 3000.0f );
   // VIEW MATRIX: specifichiamo i tre vettori per
                                          questa matrice.
   device.Transform.View = Matrix.LookAtLH( new
    Vector3( 0.0f, 3.0f,-5.0f ), new Vector3( 0.0f, 0.0f, 0.0f ),
                          new Vector3( 0.0f, 1.0f, 0.0f ) );
   // PROJECTION MATRIX: usiamo un FoV di p-greco
                                  quarti (scelta comune),
   // un aspect ratio che indichi l'uso di un quadrato
                    (altezza / larghezza = 1), e i due piani di
   //  clipping posizionati ad una distanza di 1 e 100.
   device.Transform.Projection = Matrix.PerspectiveFovLH(
                  (float)Math.PI / 4, 1.0f, 1.0f, 100.0f ); }
```

È infine anche importante impostare alcune proprietà della classe *RenderStates,* cosa che faremo dentro la funzione *OnResetDevice():*

```
public void OnResetDevice(object sender, EventArgs e)
{   Device dev = (Device)sender;
    // Abilitiamo lo ZBuffer
    device.RenderState.ZBufferEnable = true;
    // Disabilitiamo il Culling
    dev.RenderState.CullMode = Cull.None;
    // Disabilitiamo le luci: i vertici hanno già i loro colori
    dev.RenderState.Lighting = false;
}
```

Prima di tutto impostiamo lo *ZBuffer* (come visto nel primo articolo, per disegnare oggetti 3D in DirectX bisogna settare la proprietà *ZBufferEnable* del device su true per abilitare la profondità della superficie di *Rendering*). Abbiamo poi inoltre disabilitato il culling e le luci: queste impostazioni fanno in modo che un oggetto 3D venga disegnato interamente (tutte le sue facce), e che avendo già dei propri colori e non necessitando quindi di illuminazione aggiuntiva, sia rappresentato così com'è. Vedremo più approfonditamente queste proprietà nel prossimo articolo, per ora prendiamole per buone e gustiamoci i risultati che abbiamo raggiunto attraverso le screenshot in Fig. 8 del progetto di questo articolo. Il codice affrontato in questo articolo è disponibile in formato zip nel CD allegato alla rivista. Il progetto è creato con Visual Studio .NET 2003, in caso non disponiate di quest'ultima versione potete tranquillamente prendere da questo zip i file *.cs* e creare un progetto con Visual Studio .NET 1. Noterete che il codice è leggermente diverso (e più completo rispetto alle estrapolazioni che abbiamo messo qui), ma la struttura della parte DirectX, che è quella che a noi interessa, è l'esatta copia. La novità principale stà nel fatto che invece di aver disegnato un solo triangolo (come la volta precedente) abbiamo rappresentato una piramide usando quindi un totale di 6 triangoli (due per la base quadrata, e quattro per i lati obliqui). Sono certo che troverete questa parte semplice ed interessante: sono i nostri primi passi nella navigazione degli spazi 3D e nella scoperta degli oggetti a tre dimensioni.

CONCLUSIONI

In questo articolo abbiamo imparato molto: dopo aver disegnato degli oggetti nello spazio, sappiamo ora anche come muoverli ed animarli. Ci accorgiamo subito però che questa grafica è un po' irreale, troppo astratta… Nel prossimo articolo impareremo come aumentare il realismo attraverso l'uso delle luci e dei materiali.

Carlo Federico Zoffoli

Ancora OOP, interfacce ed eccezioni

Corso di Object Pascal

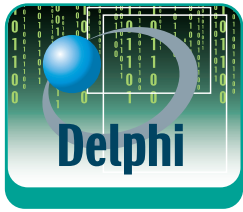
parte ultima

In questa puntata dedicata a Delphi, concluderemo con un approfondimento sull'ereditarietà e la nozione di interfaccia. Infine gestiremo gli errori con le eccezioni.

Siamo giunti al nostro ultimo appuntamento con questo corso base sul linguaggio di programmazione di Delphi. Nella puntata scorsa avevamo introdotto una serie di concetti nuovi relativi all'OOP: si è parlato di ereditarietà, cioè della possibilità da parte di una nuova classe (la classe derivata) di estendere il comportamento di una classe già esistente (la classe madre). Abbiamo poi introdotto la caratteristica del polimorfismo, cioè la capacità, oltre che di estendere, anche di alterare il comportamento di alcuni metodi della classe madre, tramite l'overriding (nel caso del binding dinamico) o l'hiding (se è binding statico). Se vi ricordate, avevamo elencato due possibili modificatori per indicare l'ambito di utilizzabilità di una proprietà, di un metodo, o di una variabile di una classe. Il modificatore *public* stava ad indicare che qualunque codice dove fosse possibile usare la classe aveva accesso a quel membro, mentre *private* indicava che solamente il codice all'interno dello stesso modulo Pascal (file, in sostanza) poteva accedere alla proprietà o metodo. Fermo restando quanto detto finora, adesso che conosciamo la possibilità di derivare classi nuove da classi esistenti, possiamo anche parlare di un terzo ambito di visibilità detto *protected*.

LA VISIBILITÀ PROTECTED

La distinzione tra *public* e *private* serve per mettere in atto la caratteristica dell'incapsulamento, di cui avevamo già parlato nel quinto articolo dedicato a questo corso: grazie alla possibilità di avere dei membri privati, il programmatore di una classe può arrogarsi il diritto di nascondere l'implementazione delle funzionalità dell'oggetto a chi lo utilizzerà, esponendo solo quelle caratteristiche che vengono offerte come pubbliche. Ricorderete, dal paragrafo precedente, che tutto il codice presente nello stesso modulo avrà la facoltà di accedere ai membri privati, ma nessun altro! Nei confronti dell'ereditarietà, però, si viene a porre il seguente problema: è vero che dati e metodi privati s'intendono interni e dunque non mi interessa renderli visibili fuori dal modulo, ma se una classe eredita da un'altra potrebbe essere utile renderle disponibili anche i meccanismi interni di elaborazione (dati e metodi). Ecco che entra in gioco il nuovo livello di visibilità *protected*, che implica, in sostanza, la stessa esposizione all'esterno di *private* (quindi solo nel modulo), ma of-

Fig. 1: *Uno scenario esemplificativo dei vari livelli di visibilità.*

fre in più anche la visibilità a tutte le classi derivate (nel modulo e fuori). Diamo un'occhiata allo scenario presentato nella Fig. 1. Abbiamo una classe chiamata *TMiaClasse* che espone un metodo ed una proprietà pubblici, rispettivamente *MetodoUno* e *MiaProp,* una variabile privata detta *dato1,* ed una *protetta* di nome *dato2.* A questo punto, creiamo una classe derivata nello stesso modulo (*TMiaDerivata*): questa avrà visibilità di tutti i membri della classe madre, compresi quelli protetti e privati. In realtà, non cambierebbe nulla anche se non fosse una derivata, dato che il motivo per cui ha visibilità totale è perché fa parte dello stesso modulo.
Ma veniamo ora alle altre due classi, definite questa volta in una unità separata *(Unit2.pas):* la classe *TAltraClasse* non deriva da *TMiaClasse* e di conseguenza potrà vedere – come abbiamo già studiato in passato – solamente i membri pubblici di *TMiaClasse* (miaclasse nell'esempio rappresenta un'ipotetica variabile di tipo *TMiaClasse* precedentemente inizializzata). Il secondo tipo (*TAltraDerivata*) presente nel secondo modulo, però, è derivato da *TMiaClasse:* questo gli dà visibilità anche del membro *dato2,* definito come *protected.* Resta comunque irraggiungibile la variabile privata *dato1*. Nella figura vedete evidenziate in rosso grassetto le righe che referenziano dati o metodi non visibili. Quando usare *private* e *protected?* Qual è l'utilità di *protected?*

NOTA

INTERFACCE E TPERSISTENTOBJECT

Normalmente quando si crea una classe che implementa un'interfaccia, come classe madre si indica *TPersistentObject* od una sua derivata. Questo perché tutte le interfacce derivano dall'interfaccia IInterface, la quale prevede tre metodi che sono già implementati in *TPersistentObject*, sollevando così il programmatore dal doverli ridefinire, trattandosi tra l'altro di funzionalità sistemistica legata alla gestione dei riferimenti agli oggetti.

LE INTERFACCE

L'ultimo argomento che abbiamo trattato la volta scorsa era relativo ai metodi *abstract*, che si possono definire in alcune classi per evitare di dare un'implementazione a funzionalità che assumono senso solo in specializzazioni della classe stessa. Ripensando all'esempio sulla geometria che stiamo portando avanti, se è vero che può essere utile definire una classe generica *TFigura,* perché ci sono delle caratteristiche comuni a tutte le figure geometriche che così dichiariamo una volta per tutte, è anche vero che un probabile metodo *Disegna* di tale classe (è normale che una figura geometrica si possa disegnare!) non può essere implementato se non nelle derivate che rappresentano figure più specifiche (ad esempio *TRettangolo* o *TEllisse*). Ora, se prediamo il significato del concetto *abstract* legato ad un metodo e lo estendiamo a tutto ciò che la classe espone, arriviamo vicini all'idea di interfaccia *(interface).*
Essa è – in senso molto lato – una classe che ha solo metodi e proprietà astratti. Notate che mancando interamente di implementazione, un'interfaccia non potrà avere variabili membro! Una classe che implementa (*implements*) un'interfaccia si obbliga a dare una definizione a tutti i metodi e tutte le proprietà che tale interfaccia richiede. La sintassi per dichiarare un'interfaccia è la seguente, tratta dall'applicazione d'esempio allegata all'articolo (che vedete in esecuzione in Fig. 2):

```
IDisegnabile = interface
  procedure Disegna(g: TCanvas); overload;
  procedure Disegna(g: TCanvas; inizio: TPunto); overload;
  procedure Disegna(g: TCanvas; x: Integer; y:
                                    Integer); overload;
end;
```

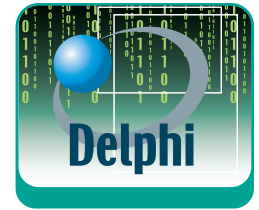
Mentre, per affermare di voler implementare un'interfaccia, una classe deve aggiungere tale interfaccia (o più di una volendo) al nome della classe madre, in questo modo:

```
TImmagine = class(TInterfacedObject, IDisegnabile)
  private
    picture: TBitmap;
  public
    constructor Create(filename: String);
    procedure Disegna(gdc: TCanvas); overload;
    procedure Disegna(gdc: TCanvas; inizio: TPunto);
                                             overload;
    procedure Disegna(gdc: TCanvas; x: Integer; y:
                                    Integer); overload;
end;
```

Fig. 2: ***L'applicazione d'esempio in esecuzione.***

Dovete pensare ad un'interfaccia come ad un obbligo, che viene assunto da chi la implementa, di dare funzionalità ai metodi e proprietà dichiarati. Ma a che ci serve dunque l'interfaccia e in che senso è diversa da una classe astratta?

IMPLEMENTARE ED EREDITARE

Cominciamo dalla seconda domanda. Estendere una classe con metodi astratti significa sì impegnarsi ad implementare tali metodi, ma fondamentalmente significa ereditare tutta la funzionalità che tale classe mi mette a disposizione. Prendiamo il caso di *TFigura* che riporto qui sotto:

```
TFigura = class(TInterfacedObject, IDisegnabile)
  private
    _useDefColor: Boolean;
    _useDefSpessore: Boolean;
    _color: TColor;
    _spessore: Integer;
    _riempi: Boolean;
    procedure _SetSpessore(valore: Integer);
    procedure _SetColore(colore: TColor);
  protected
    function _CalcolaPerimetro: Extended; virtual; abstract;
    function _CalcolaArea: Extended; virtual; abstract;
    procedure _DisegnaFigura(gdc: TCanvas; inizio:
                                TPunto); virtual; abstract;
  public
    property Colore: TColor read _color write _SetColore;
    property Spessore: Integer read _spessore write
                                                _SetSpessore;
    property Riempi: Boolean read _riempi write _riempi;
    property Perimetro: Extended read _CalcolaPerimetro;
    property Area: Extended read _CalcolaArea;
    procedure Disegna(gdc: TCanvas); overload;
    procedure Disegna(gdc: TCanvas; inizio: TPunto);
                                                  overload;
    procedure Disegna(gdc: TCanvas; x: Integer; y:
                                      Integer); overload;
end;
```

è vero che la definizione dell'argomento di disegno è lasciato alle sottoclassi (metodo astratto *_DisegnaFigura*), ma è altresì vero che tutta la funzionalità di gestione di spessore, colore e riempimento è portata a termine all'interno di questa classe generica. *TFigura* non è quindi solo un contratto da rispettare, bensì un bagaglio di codice già scritto che noi riutilizziamo nel definire, per esempio, *TRettangolo, TEllisse,* etc. Se però pensiamo al fatto che *TFigura* implementa *IDisegnabile,* qui non possiamo più dire che questa caratteristica offra funzionalità ereditata a *TFigura:* semplicemente, la classe sta dichiarando al mondo il proprio impegno a ridefinire e dare un'implementazione ai metodi che *IDisegnabile* prevede. Non ci resta che tornare alla prima questione: a che pro? Nello scrivere il codice, l'interfaccia ci torna comodo come segnaposto per tutte quelle classi – ignote in fase di compilazione – che la implementano. Dichiarando una variabile di tipo *IDisegnabile,* io potrò assegnare a tale variabile un oggetto di tipo *TImmagine* o un qualsiasi oggetto derivato da *TFigura.* Ecco così che nell'applicazione di questo mese dichiaro l'array che contiene tutto quello che l'utente chiede di disegnare sulla finestra:

```
figures: array of IDisegnabile;
```

e il metodo di risposta all'evento *OnPaint*, che disegna gli oggetti di tale array, si baserà sulla dichiarazione di *IDisegnabile* per invocare il metodo *Disegna*, sicuro del fatto che è disponibile in tutti gli oggetti che implementano tale interfaccia:

```
procedure TForm1.FormPaint(Sender: TObject);
var
  i: Integer;
begin
  for i:= 0 to Length(figures) - 1 do
  begin
    figures[i].Disegna(Canvas,origins[i]);
  end;
end;
```

Per un esempio un po' più completo delle possibilità di assegnazione ad una variabile dichiarata come interfaccia fanno testo i metodi *NuovaImmagine1Click* e *NuovaFigura1Click*. Questi vengono invocati in risposta alla selezione dei menu *File->Nuova Immagine* e *File->Nuova Figura*, e il loro obiettivo è di creare un oggetto *TImmagine* con il bitmap richiesto (NB solo i file BMP sono supportati!) o un'istanza di una derivata di *TFigura* in base al form compilato dall'utente (vedi le Fig. 3, 4 e 5).

Fig. 3: **La dialog box usata da NuovaFigura1Click in modalità Ellisse.**

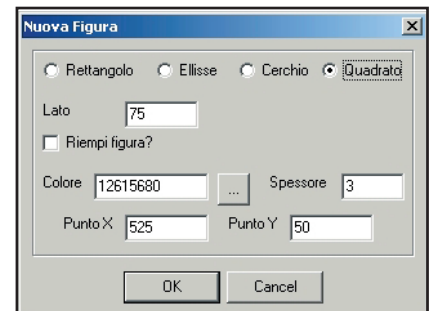



Fig. 4: **La dialog box usata da NuovaFigura1Click in modalità Quadrato.**

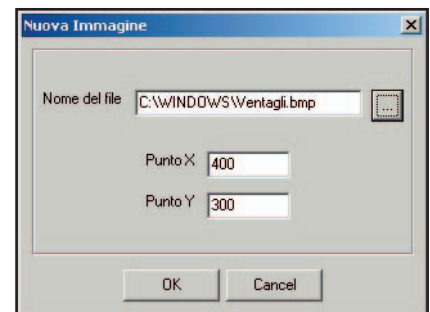



Fig. 5: **La dialog box usata da NuovaImmagine1Click.**

LE ECCEZIONI

Il codice dei due metodi appena menzionati ci porta dritti al nostro nuovo (ed ultimo) argomento. Avrete certamente notato una parola chiave *(try)* di cui ancora non abbiamo detto nulla, ma che ricorderà qualcosa a chi ha già programmato in C++ o Java (lo stesso costrutto è oggi presente anche in VB.NET e C#). Si tratta di uno statement con cui si mette in moto il meccanismo di gestione delle eccezioni *(exception handling)*. Le eccezioni sono delle condizioni anomale che si verificano durante l'esecuzione dei nostri programmi. Sono determinate normalmente da errori causati da input errati, cattivo uso delle classi di cui ci si è avvalsi, o semplicemente da uno stato del sistema inadeguato agli obiettivi del programma (memoria insufficiente, disco o dispositivo non presente, e via dicendo). Il verificarsi di una di queste eccezioni, normalmente, termina il programma in esecuzione, ritornando il controllo al sistema operativo: se però, prevedendo quali tipi di eccezioni

potrebbero aver luogo, decidiamo di gestire noi l'eventuale problema e di permettere così al nostro software di procedere rimediando all'anomalia, allora dobbiamo includere le istruzioni a fronte delle quali possono insorgere le difficoltà in un blocco *try/except*, come segue

```
try
  // istruzioni potenzialmente problematiche
except
  // gestione delle eccezioni, se presenti
end;
```

La sezione *except* contiene una struttura sintattica simile al *case*, con una serie di on seguiti dal tipo di eccezione che intendiamo gestire nella sezione relativa.

```
try
except
  on EZeroDivide do GestisciDivisionePerZero;
  on EOverflow do GestisciOverflow;
  on EMathError do GestisciMathErrorGenerico;
end;
```

Sappiate che, di fatto, le eccezioni non sono altro che istanze di classi che derivano da *Exception*.
Anche noi possiamo, in maniera molto semplice, creare delle eccezioni personalizzate

```
EGeometryException = class(Exception);
```

Anche se la mia eccezione in questo caso non aggiunge nulla di nuovo alla funzionalità di base di *Exception*. La definizione qui sopra mi permette di gestire in una sezione on separata le anomalie riscontrate e registrate specificatamente dalle mie classi. Quando stiamo disegnando una libreria ad oggetti, possiamo infatti decidere di gestire determinate condizioni d'errore attraverso il lancio di eccezioni verso il codice che invoca le nostre classi. Partendo sempre dall'applicativo esemplificativo sulle figure geometriche, lo spessore minore di 1, o una base, altezza o raggio minori o uguali a 0 sono problemi su cui il codice di implementazione interno della classe non ha modo di intervenire se non avvertendo l'utente trasmettendogli un'eccezione. Se il tipo d'eccezione che lancio è un tipo mio specifico, potrò conseguentemente gestire la situazione in maniera più consona e particolareggiata in un blocco on dedicato al mio tipo di oggetto derivato da *Exception*. Qui vedete come la classe *TRettangolo* lancia un'eccezione se la base viene impostata su un valore non idoneo

```
if n > 0 then
  b:= n
else
  raise EGeometryException.Create('La base deve
                                        essere > 0');
```

Osservate anche l'interessante caso del costruttore di TImmagine

```
constructor TImmagine.Create(filename: String);
begin
  try
    picture:= TBitmap.Create();
    picture.LoadFromFile(filename);
  except
    on e: Exception do raise
              EGeometryException.Create(e.Message);
  end;
end;
```

in cui noi gestiamo un'eccezione generica lanciandone un'altra nostra. Notate la sintassi alternativa utilizzata con lo statement *on*, in cui l'oggetto di eccezione (di tipo *Exception*) viene assegnato ad una variabile *(e)*, per poterne poi utilizzare proprietà e metodi (nell'esempio viene richiamata la proprieta *e.Message* che restituisce il messaggio di descrizione del problema). Infine, il form principale del programma d'esempio si premerà di catturare le eventuali eccezioni *EGeometryException* lanciate dalla libreria di figure geometriche e di gestirle invitando l'utente ad immettere dei valori corretti negli appositi form (vedi Fig. 6).

Fig. 6: ***La libreria di classi lancia un'eccezione e il form principale la cattura.***

CONCLUSIONI

Giunti fino qui, dovreste avere una visione abbastanza completa del linguaggio Object Pascal. A corredo di questo articolo trovate una versione evoluta dell'applicazione per disegnare figure geometriche su un canvas. Questa versione definisce un'interfaccia di base *IDisegnabile*, implementata da *TImmagine* (per disegnare file bitmap) e *TFigura* (classe astratta che fa da base per *TRettangolo* e *TEllisse* – a loro volta, da *TRettangolo* deriva *TQuadrato* e da *TEllisse* deriva *TCerchio*). Nel codice, trovate esempi di lancio di eccezioni (con *raise*) e cattura delle stesse (*try/except,*), così da avere un esempio pratico di quanto visto in questo ultimo appuntamento. Per poter apprendere davvero bene un linguaggio di programmazione è necessaria molta pratica ed esperienza: l'unico modo per ottenerle entrambe è scrivere codice, per cui non abbiate paura di mettervi subito al lavoro e realizzare le vostre idee sfruttando le conoscenze di Object Pascal che avete acquisito. Nel box a lato trovate un paio di link utili come ispirazione per approfondire e fare, mentre per piccoli dubbi e quesiti potete provare a scrivermi all'e-mail *federico.mestrone@ioprogrammo.it*, pazientando un po' per la risposta perché tempo libero è pochissimo! Buon proseguimento.

Federico Mestrone

SUL WEB

Ecco alcuni siti che trattano di programmazione sotto Delphi:

come introduzione al linguaggio
www.delphibasics.co.uk/

mentre per librerie, tips & tricks, codice sorgente, articoli
http://delphi.about.com/
e
www.delphidabbler.com/

La tecnologia GDI+ del Framework .NET

Grafica in VB.NET

Continua il viaggio all'interno della tecnologia GDI+. Realizzeremo disegni e immagini con le problematiche legate al sistema di coordinate.

Nel precedente articolo abbiamo introdotto la tecnologia GDI+ ed abbiamo visto come la prima operazione da compiere, per disegnare elementi grafici in una qualsiasi periferica di visualizzazione, consiste nell'ottenere un riferimento ad un oggetto *Graphics*. Un oggetto *Graphics* rappresenta, in pratica, una superficie di disegno (non deve essere necessariamente una superficie visibile), normalmente l'area client di un oggetto *Form*. Tutti gli esempi di codice descritti nel precedente articolo assumevano che l'origine delle coordinate fosse nell'angolo superiore sinistro dell'area client, e che tutti i valori fossero espressi in pixel. L'oggetto *Graphics* consente di modificare le impostazioni predefinite permettendo di cambiare il sistema di coordinate. La seconda parte dell'articolo sarà incentrata sulla gestione delle immagini di tipo raster e vettoriali.

I SISTEMI DI COORDINATE

In GDI+, quando si disegna un oggetto grafico (incluse immagini e testo), vengono utilizzati tre differenti sistemi di coordinate: di ambiente, di pagina e di periferica, vediamoli in dettaglio.

Il sistema di coordinate di ambiente (o complessivo) è il sistema utilizzato di default per modellare l'ambiente grafico. Tutte le coordinate che vengono passate ai metodi grafici sono espresse nel sistema di coordinate di ambiente.

Il sistema di coordinate della pagina è il sistema utilizzato dalla superficie su cui si disegna: sia essa una form, un controllo o un documento da stampare. Di solito, l'origine e la scala di questo sistema coincidono con quelle del sistema di coordinate d'ambiente, ma questo non è obbligatorio. È possibile fissare l'origine del sistema in un punto qualsiasi della form. La conversione tra le coordinate dell'ambiente e quelle della pagina viene definita trasformazione d'ambiente.

Il sistema di coordinate della periferica è il sistema utilizzato dalla periferica fisica sulla quale avviene l'operazione di disegno. Se si disegna su schermo la periferica è la finestra, se l'output viene inviato alla stampante la periferica diventa il foglio. È possibile definire con precisione l'unità di misura del sistema (pollici,millimetri, ecc), il rapporto tra l'asse verticale e quello orizzontale. La conversione tra le coordinate della pagina e quelle della periferica viene definita trasformazione di pagina.

Supponiamo di voler disegnare una linea retta, che colleghi due punti specificati da due coppie di coordinate, utilizzando il metodo DrawLine descritto nell'articolo precedente. In particolare vogliamo disegnare una linea di colore rosso che colleghi i punti di coordinate (1,1) e (100,100), il codice necessario è il seguente:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
Dim OggettoPen = New Pen(Color.Red)
OggettoGrafico.DrawLine(OggettoPen, 1, 1, 100, 100)
OggettoGrafico.Dispose()
```

Fig. 1: Linea nel sistema di coordinate di ambiente.

Con l'istruzione *OggettoGrafico.DrawLine(OggettoPen*, 1, 1, 100, 100), i punti passati al metodo *DrawLine* si trovano nello spazio di coordinate complessivo, per cui il risultato è quello riportato in Fig. 1. Facciamo ora l'ipotesi di voler fare riferimento ad un sistema di coordinate la cui origine si trovi in un punto all'interno dell'area client, posto, per esempio, ad 80 pixel di distanza dal margine sinistro ed a 70 pixel di distanza dall'estremità superiore. Tra i metodi che l'oggetto *Graphics* mette a disposizione per influire sulle modalità della trasformazione d'ambiente, possiamo utilizzare il metodo *TranslateTransform*. Il metodo *TranslateTransform* permette di modificare l'origine del sistema di coordinate spostandola nei punti specificati come argomento. In questo modo, per disegnare lo stesso oggetto in una posizione differente, sarà sufficiente traslare l'origine degli assi *X-Y* di 80 unità nella direzione *x* e 70 unità nella direzione *y*:

```
OggettoGrafico.TranslateTransform(80, 70)
```

Fig. 2: Linea nel sistema di coordinate di pagina

NOTA

Una immagine bitmap è costituita da una matrice di bit che permette di specificare il colore di ogni pixel in una matrice rettangolare di pixel. Il numero di colori che è possibile assegnare ad ogni pixel è determinato dal numero di bit destinati al singolo pixel. Se ad esempio ogni pixel è rappresentato da 4 bit, sarà possibile assegnare ad un dato pixel uno dei 16 colori diversi disponibili (2^4 = 16). Il formato BMP è un formato standard utilizzato da Windows per la memorizzazione di immagini indipendenti da periferiche e da applicazioni. Il numero di bit per pixel (1, 4, 8, 15, 24, 32 o 64) per un dato file BMP viene specificato nell'intestazione del file. I file BMP più comuni utilizzano 24 bit per pixel.

```
OggettoGrafico.DrawLine(OggettoPen, 1, 1, 100, 100)
```

Il metodo *RotateTransform* permette di ruotare di un angolo qualsiasi, tutto ciò che viene disegnato sull'oggetto *Graphics*. Vogliamo, ad esempio, disegnare un rettangolo, utilizzando il metodo *DrawRectangle* ed applicargli una rotazione di 45 gradi, il codice sarà il seguente:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
Dim OggettoPen = New Pen(Color.Red)
'applico la rotazione di 45 gradi
OggettoGrafico.RotateTransform(45)
OggettoGrafico.DrawRectangle(OggettoPen, 100, 10, 100, 50)
OggettoGrafico.Dispose()
```

Utilizzando il metodo *ScaleTransform,* è possibile modificare la scala utilizzata, lungo i due assi, per disegnare sull'oggetto *Graphics*. I fattori di scala in direzione *x* ed *y* devono essere passati come argomento. Fino a quando utilizziamo il pixel come unità di misura, le coordinate della periferica corrispondono alle coordinate della pagina. Per applicare una trasformazione di pagina, la classe *Graphics* mette a disposizione le proprietà *PageUnit* e *PageScale*. La proprietà PageUnit accetta un valore enumerato *GraphicsUnit* per specificare un'unità di misura diversa dal pixel (valore di default per il video). Sono ammessi i valori:

- **Inch,** indica come unità di misura il pollice.
- **Millimiter,** indica come unità di misura il millimetro.
- **Point,** indica come unità di misura un punto della stampante, 1/72 di pollice.
- **Display,** indica come unità di misura 1/75 di pollice.
- **Document,** indica come unità di misura quella del documento, 1/300 di pollice:

Vogliamo, ad esempio, disegnare una linea che colleghi i punti di coordinate (0,0) e (20,50), dove il punto (20,50) si trova 20 millimetri a destra e 50 millimetri in basso rispetto al punto (0, 0), dovremo quindi scrivere:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
Dim OggettoPen = New Pen(Color.Red)
OggettoGrafico.PageUnit = GraphicsUnit.Millimeter
OggettoGrafico.DrawLine(OggettoPen, 0, 0, 20, 50)
OggettoGrafico.Dispose()
```

Dopo aver eseguito questo codice, possiamo notare come lo spessore della linea disegnata sia aumentato. Questo effetto si ottiene poiché gli oggetti Pen (per default) hanno una larghezza di una unità, misurata rispetto alle coordinate della periferica. Per questo, adottando i millimetri come unità di misura, la linea è stata disegnata con un oggetto Pen largo un millimetro. Se vogliamo evitare questo effetto possiamo utilizzare le proprietà di sola lettura *DpiX* e *DpiY* che permettono di ottenere la risoluzione orizzontale e verticale, in modo da determinare il valore da passare al costruttore dell'oggetto *Pen:*

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
OggettoGrafico.PageUnit = GraphicsUnit.Millimeter
Dim OggettoPen = New Pen(Color.Red, 1 /
                                        OggettoGrafico.DpiX)
OggettoGrafico.DrawLine(OggettoPen, 0, 0, 20, 50)
OggettoGrafico.Dispose()
```

Infine, la proprietà *PageScale* permette di ridurre o ingrandire di un dato fattore le unità di misura globali e quelle della pagina.

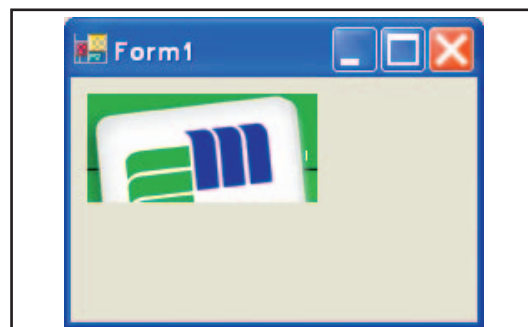



Fig. 3: Immagine clonata.

GESTIONE DELLE IMMAGINI

La tecnologia GDI+ prevede un sottoinsieme che è stato creato specificamente per mostrare ed elaborare tanto immagini raster *(bitmap)* che immagini vettoriali *(metafile)*. I due oggetti più rilevanti per l'elaborazione delle immagini sono:

- La classe **Image,** classe base astratta, che fornisce metodi per salvare e caricare immagini da disco, ed altre operazioni sulle immagini
- La classe **Bitmap**, che deriva da Image, e consente di espanderne le funzionalità fornendo metodi aggiuntivi per modificare le immagini raster e per accedere ai singoli pixel dell'immagine. La classe *Bitmap* supporta numerosi formati di file, compresi BMP, GIF, JPEG, PNG e TIFF.
- La classe **Metafile** permette di espandere le funzionalità della classe *Image* fornendo metodi aggiuntivi per salvare, caricare, modificare ed esaminare immagini di tipo vettoriale (memorizzate sotto forma di comandi e impostazioni di disegno). La classe metafile supporta immagini memorizzate nei formati: *WMF(Windows Metafile), EMF(Enhanced Metafile), EMF+*

In definitiva, per caricare e visualizzare un'immagi-

ne vettoriale, sono necessari un oggetto *Graphics* ed un oggetto *Metafile*. Per visualizzare un'immagine raster, sono necessari un oggetto *Graphics* ed un oggetto *Bitmap*. Le classi *Image* e *Bitmap* appartengono al namespace *System.Drawing,* mentre la classe *Metafile* appartiene al namespace *System.Drawing.Imaging.*

IL METODO DRAWIMAGE

L'oggetto *Graphics* mette a disposizione il metodo *DrawImage,* che permette di visualizzare un'immagine ricevuta da un oggetto *Metafile* o *Bitmap* passato come argomento. Esistono numerose versioni di *overload* (trenta) del metodo *DrawImage,* quindi è possibile fornire argomenti in svariati modi. La versione più semplice, accetta come argomenti l'immagine da mostrare a video e le coordinate dell'angolo superiore sinistro dell'area di destinazione. La prima operazione da compiere è quella di caricare un'immagine in un oggetto *Bitmap* (o *Metafile*), per questo possiamo utilizzare il relativo costruttore e scrivere:

```
Dim OggettoBitmap As New Bitmap("C:\MiaImmagine.jpg")
```

In cui *C:\MiaImmagine.jpg* è il percorso completo del file contenente l'immagine da mostrare. Dopo la creazione dell'oggetto *Bitmap,* è necessario passare questo oggetto come argomento al metodo *DrawImage* insieme alle coordinate. Per disegnare l'immagine bitmap con l'angolo superiore sinistro in corrispondenza di (50, 50) dobbiamo scrivere:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
Dim OggettoBitmap As New Bitmap("C:\MiaImmagine.jpg")
OggettoGrafico.DrawImage(OggettoBitmap, 50, 50)
OggettoBitmap.Dispose()
OggettoGrafico.Dispose()
```

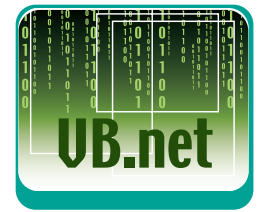
Un'altra versione del metodo *DrawImage* riceve, come argomenti, un oggetto *Bitmap* contenente l'immagine da disegnare, ed un oggetto *Rectangle,* che permette di specificare il rettangolo in cui verrà tracciata l'immagine. Se le dimensioni del rettangolo di destinazione non corrispondono alle dimensioni dell'immagine originale, l'immagine verrà ridimensionata ed adattata al rettangolo di destinazione. Sfruttando questo comportamento ed utilizzando la proprietà *Width* dell'oggetto *Bitmap* che specifica la larghezza dell'immagine, e la proprietà *Height* dell'oggetto *Bitmap* che ne specifica l'altezza, possiamo ingrandire o comprimere un'immagine. Nel codice seguente definiamo tre rettangoli di dimensioni diverse. Il primo rettangolo ha le stesse dimensioni dell'immagine, pertanto questa verrà disegnata nelle sue dimensioni reali. Il secondo rettangolo avrà le dimensioni pari alla metà dell'immagine, pertanto l'immagine risulterà compressa. Il terzo rettangolo avrà le dimensioni pari al doppio dell'immagine, pertanto l'immagine risulterà ingrandita rispetto alle dimensioni originali.

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
Dim OggettoBitmap As New Bitmap("C:\MiaImmagine.jpg")
'definisce un rettangolo delle esatte dimensioni
                                        dell'immagine
Dim RettangoloEsatto As New Rectangle(0, 0, _
      OggettoBitmap.Width, OggettoBitmap.Height)
'definisce un rettangolo delle dimensioni dimezzate
                                        dell'immagine
Dim RettangoloCompresso As New Rectangle(50, 0, _
   OggettoBitmap.Width / 2, OggettoBitmap.Height / 2)
'definisce un rettangolo delle dimensioni uguali al
                                 doppio dell'immagine
Dim RettangoloEspanso As New Rectangle(100, 0, _
   OggettoBitmap.Width * 2, OggettoBitmap.Height * 2)
OggettoGrafico.DrawImage(OggettoBitmap,
                                     RettangoloEsatto)
OggettoGrafico.DrawImage(OggettoBitmap,
                                  RettangoloCompresso)
OggettoGrafico.DrawImage(OggettoBitmap,
                                    RettangoloEspanso)
OggettoBitmap.Dispose()
OggettoGrafico.Dispose()
```

Un'ulteriore versione del metodo *DrawImage* permette di specificare, oltre al rettangolo di destinazione, anche il rettangolo di origine. Il parametro relativo al rettangolo di origine permette di determinare quale porzione dell'immagine originale dovrà essere disegnata. Utilizzando ancora una volta le proprietà *Width* ed *Height* possiamo, quindi, ritagliare una parte dell'immagine originale. Anche in questo caso, se le dimensioni del rettangolo di destinazione non corrispondono alle dimensioni del rettangolo di origine, l'immagine verrà ridimensionata e adattata al rettangolo di destinazione.

IL METODO CLONE

La classe *Bitmap* mette a disposizione il metodo *Clone,* che permette la creazione di una copia di un oggetto *Bitmap* esistente. Il metodo *Clone* riceve come parametri:

- Un oggetto *Rectangle* relativo al rettangolo di origine, che permette di determinare quale porzione della bitmap originale dovrà essere copiata.
- Un valore di tipo *PixelFormat* che definisce il formato dei dati relativi al colore per ciascun pixel dell'immagine (in particolare definisce il numero di bit di memoria associati ad ogni pixel)

GRAFICA VETTORIALE

La grafica vettoriale implica il disegno di primitive, quali linee, curve e poligoni, specificati da un insieme di punti in un sistema di coordinate. Il disegno non viene rappresentato dalla serie di pixel, che lo compone sullo schermo, ma è possibile, ad esempio, disegnare una linea retta specificando soltanto le coordinate dei due punti estremi, oppure disegnare un rettangolo tramite un punto che ne rappresenti la posizione dell'angolo superiore sinistro ed un paio di numeri che ne indicano la larghezza e l'altezza. Gli oggetti messi a disposizione da GDI+ per la gestione della grafica vettoriale sono racchiusi nel namespace *System.Drawing.Drawing2D.*

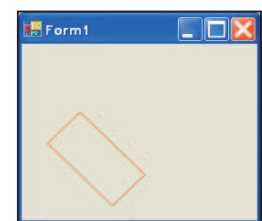
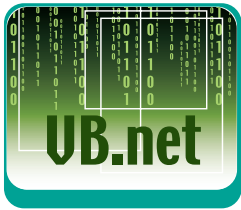



Fig. 4: Il rettangolo ruotato.

NOTA

GRAFICA RASTER (IMMAGINI)

In un'immagine raster, lo spazio è suddiviso in migliaia di quadratini detti pixel. Ogni quadratino può essere di un colore, ed è compito del programma di grafica impostare il colore di ogni quadratino in base al tipo di strumento di disegno. Le immagini di questo tipo vengono memorizzate come bitmap, in cui ogni colore dei singoli punti sullo schermo viene trasformato in numero e memorizzato in una matrice. Ogni elemento nella matrice, è accessibile tramite una coppia di coordinate x-y, che identifica il singolo pixel. Gli oggetti messi a disposizione da GDI+ per la gestione delle immagini sono racchiusi nel namespace *System.Drawing.Imaging*.

Vogliamo, ad esempio, creare un oggetto *Bitmap* mediante la clonazione della metà superiore di un oggetto *Bitmap* esistente, il codice sarà il seguente:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
        Dim OggettoBitmap As New Bitmap(
                                  "C:\MiaImmagine.jpg")
        Dim RettangoloSorgente As New Rectangle(0, 0,
            OggettoBitmap.Width, OggettoBitmap.Height / 2)
Dim OggettoBitmapClonato As Bitmap =
            OggettoBitmap.Clone(RettangoloSorgente, _
                 Drawing.Imaging.PixelFormat.DontCare)
OggettoGrafico.DrawImage(OggettoBitmapClonato, 10, 10)
OggettoBitmap.Dispose()
OggettoBitmapClonato.Dispose()
OggettoGrafico.Dispose()
```

Fig. 5: **Immagine esatta, ridotta, ingrandita.**

I METODI LOAD E SAVE

I metodi *FromFile* e *LoadFromStream* delle classi *Image* e *Bitmap* consentono di caricare un'immagine da un file o da un oggetto Stream aperto, dove per *Stream* si intende una generica sequenza di byte (flusso). Un flusso di byte non deve necessariamente provenire da un file ma ad esempio da una qualsiasi periferica di input/output, o da un socket TCP/IP. L'oggetto *Stream* permette la visualizzazione generica di questi diversi tipi di flusso, senza che lo sviluppatore venga a contatto con i dettagli specifici del sistema operativo e con le periferiche sottostanti. Scriviamo una procedura che utilizza il metodo *FromFile* ed un controllo *OpenFileDialog* per richiedere all'utente il nome dell'immagine da visualizzare ed utilizziamo il metodo *DrawImage* per mostrarla sullo schermo. Il codice da scrivere sarà il seguente

```
    Private Sub MostraImmagineDaFile()
        Dim OggettoGrafico As Graphics=Me.CreateGraphics
        OpenFileDialog1.Filter = "Image files|*.bmp;
                        *.gif;*.jpg;*.jpeg;*.tif;*.png;"
        If OpenFileDialog1.ShowDialog=DialogResult.OK Then
            Dim OggettoBmp As Bitmap = Bitmap.FromFile(
                           OpenFileDialog1.FileName)
            OggettoGrafico.DrawImage(OggettoBmp, 0, 0)
            OggettoBmp.Dispose()
            OggettoGrafico.Dispose()
        End If
    End Sub
```

Il metodo *Save* delle classi *Image* e *Bitmap* permette di salvare un'immagine memorizzata nell'oggetto corrispondente. Questo metodo accetta come argomenti: il nome del file ed il formato di destinazione. Utilizzando un controllo *SaveFileDialog* possiamo scrivere una procedura che permetta di salvare un'immagine contenuta in un oggetto *Bitmap*, chiedendo all'utente di specificare il formato desiderato.

```
Private Sub SalvaImmagine()
   Dim OggettoGrafico As Graphics = Me.CreateGraphics
   Dim OggettoBitmap As New Bitmap("C:\MiaImmagine.jpg")
   OggettoGrafico.DrawImage(OggettoBitmap, 0, 0)
     With SaveFileDialog1()
         .Title = "Seleziona il formato del file di
                                        destinazione"
         .Filter = "Bitmap|*.bmp|GIF|*.gif|JPEG|*.jpg"
         .OverwritePrompt = True
         If .ShowDialog = DialogResult.OK Then
             Select Case System.IO.Path.GetExtension(
                                     .FileName).ToUpper
                 Case ".BMP"
   OggettoBitmap.Save(.FileName,
             System.Drawing.Imaging.ImageFormat.Bmp)
   Case ".GIF"
      OggettoBitmap.Save(.FileName,
             System.Drawing.Imaging.ImageFormat.Gif)
   Case ".JPG"
      OggettoBitmap.Save(.FileName,
             System.Drawing.Imaging.ImageFormat.Jpeg)
   Case Else
      MessageBox.Show("Formato non ammesso",
                "Attenzione", MessageBoxButtons.OK,
                              MessageBoxIcon.Error)
         End Select
        End If
      End With
   OggettoBitmap.Dispose()
   OggettoGrafico.Dispose()
End Sub
```

CONCLUSIONI

In questo articolo abbiamo descritto le problematiche relative all'utilizzo dei diversi sistemi di coordinate ammessi in GDI+, ed abbiamo analizzato la gestione delle immagini. Nel prossimo articolo concluderemo l'argomento GDI+ occupandoci della gestione del testo.

Luigi Buono

Collegare uno o più metodi ad un'azione

Delegati ed eventi

In questo appuntamento ci occuperemo di due strumenti molto utili. Sia per realizzare applicazioni facilmente estensibili, sia per ottenere uno stile chiaro nella gestione delle azioni cui il software è interessato.

Con questa lezione avviamo il rush finale del corso, dedicato agli elementi più avanzati ed innovativi di C# e .NET. Cominceremo parlando dei delegati, che consentono di richiamare dei metodi il cui nome e la cui posizione è nota solo a runtime, e non al momento della stesura del codice. Proseguiremo trattando gli eventi, utili per realizzare applicazioni modulari capaci di gestire in modo chiaro e semplice ogni tipo di azione prevista dallo sviluppatore.

DELEGATI

Un delegato è un oggetto che può far riferimento ad un metodo. Chi ha qualche esperienza di programmazione con C/C++ può immaginare i delegati di C# come una sorta di puntatori a funzione. Con i delegati è possibile richiamare un metodo senza conoscerne a priori il nome. I delegati giocano un ruolo importante nello sviluppo di applicazioni che godono della possibilità di essere facilmente estese attraverso l'uso di componenti che funzionino a mo' di plug-in. Questo concetto sarà presto più chiaro, dopo aver dimostrato praticamente l'utilizzo e lo scopo dei delegati. La sintassi di riferimento per la definizione di un delegato è:

```
delegate tipo-restituito nome(lista-parametri);
```

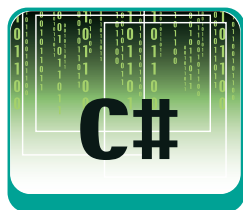
delegate è la parola chiave utilizzata per la definizione di un delegato; *tipo-restituito* è il tipo del valore di ritorno dei metodi che potenzialmente potranno essere richiamati attraverso le istanze del delegato; *nome* è il nome che il programmatore vuole assegnare al delegato; *lista-parametri* è l'elenco dei parametri accettati dai metodi che potenzialmente potranno essere richiamati attraverso le istanze del delegato. Una volta definito il delegato che si intende sfruttare, è possibile associare alle sue istanze qualsiasi metodo che ne rispetti la firma, cioè che abbia lo stesso tipo di ritorno e la stessa lista di argomenti specificati alla definizione del delegato. Ogni singolo delegato, quindi, può essere usato per rappresentare una sola famiglia di metodi. La parola *delegate* è un po' come la parola *class:* con essa si crea un modello, non un'istanza. Istanziare un delegato, poi, è come istanziare una classe:

```
NomeDelegato nomeIstanzaDelegato = new
                        NomeDelegato(nome-metodo);
```

L'argomento fornito al "costruttore" del delegato, nome-metodo, è il nome del metodo che l'istanza del delegato sarà in grado di richiamare. Naturalmente il metodo specificato deve rispettare la firma del delegato, altrimenti niente da fare. Richiamare un metodo delegato è semplice e naturale:

```
risultato = nomeIstanzaDelegato(lista-argomenti);
```

Passiamo all'analisi di un esempio pratico, che risolverà parecchi dubbi:

```
// Definizione del delegato.
delegate int OperazioneAritmetica(int operando1,
                                  int operando2);
class Test {
  static int addizione(int a, int b) {
    return a + b; }
  static int sottrazione(int a, int b) {
    return a - b;}
  static int moltiplicazione(int a, int b) {
    return a * b; }
  static int divisione(int a, int b) {
    return a / b; }
  public static void Main() {
    int o1 = 117;
    int o2 = 13;
    int risultato;
    // Istanza del delegato.
    OperazioneAritmetica calcola = new
                        OperazioneAritmetica(addizione);
    risultato = calcola(o1, o2);
    System.Console.WriteLine("Prima operazione,
                             risultato: " + risultato);
    // Nuova istanza del delegato.
    calcola = new OperazioneAritmetica(sottrazione);
```

```
    risultato = calcola(o1, o2);
    System.Console.WriteLine("Seconda operazione,
                                 risultato: " + risultato);
    // Nuova istanza del delegato.
    calcola = new OperazioneAritmetica(moltiplicazione);
    risultato = calcola(o1, o2);
    System.Console.WriteLine("Terza operazione,
                                 risultato: " + risultato);
    // Nuova istanza del delegato.
    calcola = new OperazioneAritmetica(divisione);
    risultato = calcola(o1, o2);
    System.Console.WriteLine("Quarta operazione,
                                 risultato: " + risultato); }
}
```

Questo esempio contiene la definizione di un delegato:

```
delegate int OperazioneAritmetica(int operando1,
                                  int operando2);
```

Con questa definizione sarà possibile istanziare dei delegati che "puntino" a tutti quei metodi la cui firma sia caratterizzata da un valore di ritorno di tipo int e da due parametri anch'essi di tipo int. La classe *Test*, contiene quattro metodi statici che soddisfano questo requisito: *addizione(), sottrazione(), moltiplicazione()* e *divisione()*. Nel *Main()* della classe si creano quattro differenti istanze del delegato *OperazioneAritmetica*, ognuna delle quali viene associata ad uno dei metodi statici appena presentati. I delegati ottenuti sono quindi utilizzati come degli alias verso i quattro metodi statici della classe.

OMETTIAMO I METODI STATICI

L'impiego di metodi statici, nell'esempio appena visto, non è di per sé significativo: i delegati possono far riferimento anche a metodi di altro tipo.
Rivediamo l'esempio precedente, forzando una modifica che ci consenta di trasformare *addizione(), sottrazione(), moltiplicazione()* e *divisione()* da metodi statici a comuni metodi d'istanza:

```
delegate int OperazioneAritmetica(int operando1,
                                  int operando2);
class Aritmetica {
  public int addizione(int a, int b) {
    return a + b; }
  public int sottrazione(int a, int b) {
    return a - b;}
  public int moltiplicazione(int a, int b) {
    return a * b;}
  public int divisione(int a, int b) {
    return a / b;} }
class Test {
 public static void Main() {
  Aritmetica a = new Aritmetica();
  int o1 = 117;
  int o2 = 13;
  int risultato;
  OperazioneAritmetica calcola = new
                  OperazioneAritmetica(a.addizione);
  risultato = calcola(o1, o2);
  System.Console.WriteLine("Prima operazione,
                              risultato: " + risultato);
  calcola = new OperazioneAritmetica(a.sottrazione);
  risultato = calcola(o1, o2);
  System.Console.WriteLine("Seconda operazione,
                              risultato: " + risultato);
  calcola = new OperazioneAritmetica(a.moltiplicazione);
  risultato = calcola(o1, o2);
  System.Console.WriteLine("Terza operazione,
                              risultato: " + risultato);
  calcola = new OperazioneAritmetica(a.divisione);
  risultato = calcola(o1, o2);
  System.Console.WriteLine("Quarta operazione,
                              risultato: " + risultato); }
}
```

I quattro metodi sono stati raccolti all'interno della classe *Aritmetica*, che è necessario istanziare per avere accesso alle sue funzionalità.

MULTICASTING DEI DELEGATI

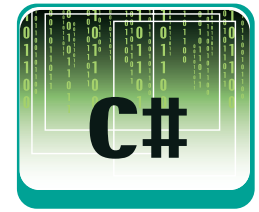
Si chiama muticasting un'interessantissima proprietà dei delegati, che permette di concatenare l'uso di più metodi in una sola istanza di un delegato. In pratica, è possibile, con una sola chiamata ad un delegato, eseguire *n* metodi con la stessa firma, sullo stesso gruppo di argomenti. Farlo è molto semplice: basta "sommare" tra di loro più istanze di uno stesso delegato, con l'operatore + (o con +=). Uno specifico metodo, poi, può essere eliminato dalla catena usando l'operatore – (o –=). C'è una sola restrizione: possono partecipare ad un multicasting solo quei metodi il cui tipo di ritorno sia *void*.
Analizziamo il seguente esempio:

```
delegate void OperazioneAritmetica(int operando1,
                                   int operando2);
class Test {
  static void addizione(int a, int b) {
    int ris = a + b;
    System.Console.WriteLine(a + " + " + b + " = " + ris); }
  static void sottrazione(int a, int b) {
    int ris = a - b;
    System.Console.WriteLine(a + " - " + b + " = " + ris); }
  static void moltiplicazione(int a, int b) {
    int ris = a * b;
    System.Console.WriteLine(a + " * " + b + " = " + ris); }
```

BIBLIOGRAFIA

• **GUIDA A C#**
Herbert Schildt
(McGraw-Hill)
2002
ISBN 88-386-4264-8

• **INTRODUZIONE A C#**
Eric Gunnerson
(Mondadori Informatica)
2001
ISBN 88-8331-185-X

• **C# GUIDA PER LO SVILUPPATORE**
Simon Robinson e altri
(Hoepli)
2001
ISBN 88-203-2962-X

```
  static void divisione(int a, int b) {
    int ris = a / b;
    System.Console.WriteLine(a + " / " + b + " = " + ris); }
  public static void Main() {
    OperazioneAritmetica calcola = new
                             OperazioneAritmetica(addizione);
    calcola += new OperazioneAritmetica(sottrazione);
    calcola += new OperazioneAritmetica(moltiplicazione);
    calcola += new OperazioneAritmetica(divisione);
    calcola(10, 5);}
}
```

Siamo di fronte ad una variante del primo esempio dimostrato nel paragrafo precedente. Anzitutto, nella definizione del delegato, il tipo di ritorno è stato cambiato da *int* a *void*. Poi sono stati modificati i quattro metodi statici *addizione()*, *sottrazione()*, *moltiplicazione()* e *divisione()*. Ora, il risultato calcolato da questi quattro metodi non è più restituito al codice chiamante (il tipo di ritorno è diventato *void*), ma vengono direttamente stampati sullo schermo. Nel *Main()* dell'applicazione sono stati istanziati quattro delegati di tipo *OperazioneAritmetica*, uno per ogni metodo statico previsto, ed insieme sono stati concatenati all'interno del riferimento *calcola*. Richiamando il delegato *calcola()* su due operandi qualsiasi si ottiene la chiamata ordinata dei quattro metodi statici elaborati in precedenza. L'output mostrato è:

10 + 5 = 15
10 - 5 = 5
*10 * 5 = 50*
10 / 5 = 2

Con il multicasting dei delegati si soffre della limitazione di non poter usare altro tipo di ritorno all'infuori di *void*. Come fare, allora, se si desidera calcolare un risultato che, in qualche modo, possa poi essere successivamente manipolato dal programmatore? E' sufficiente introdurre nella definizione del delegato (e dei metodi che dovranno essere riferiti) almeno un parametro sul quale si faccia effetto collaterale, come si dice in gergo, cioè il cui contenuto venga modificato dai singoli metodi richiamati. Lo si può fare con un oggetto, oppure passando per riferimento (parola chiave *ref*) uno dei tipi che solitamente viene passato per valore. Ecco un esempio:

```
delegate void OperazioneAritmetica(int operando1,
                              int operando2, ref int risultato);
class Test {
  static void addizione(int a, int b, ref int ris) {
    ris += a + b;}
  static void sottrazione(int a, int b, ref int ris) {
    ris += a - b; }
  static void moltiplicazione(int a, int b, ref int ris) {
    ris += a * b; }
  static void divisione(int a, int b, ref int ris) {
    ris += a / b; }
  public static void Main() {
    OperazioneAritmetica calcola =
                   new OperazioneAritmetica(addizione);
    calcola += new OperazioneAritmetica(sottrazione);
    calcola += new OperazioneAritmetica(moltiplicazione);
    calcola += new OperazioneAritmetica(divisione);
    int ris = 0;
    calcola(10, 5, ref ris);
    System.Console.WriteLine(ris);}
}
```

Con la nuova modifica apportata, lo scopo di ogni singolo metodo diventa eseguire il proprio calcolo per poi sommarlo al valore già contenuto nella variabile ris passata per riferimento. In questa maniera è stato svolto il calcolo:

*(10 + 5) + (10 – 5) + (10 * 5) + (10 / 5)*

INTRODUZIONE AGLI EVENTI

Gli eventi ricoprono un importante ruolo nei più moderni stili di programmazione. Cerchiamo anzitutto di inquadrare cosa sia un evento. Un evento è la notifica dell'accadimento di un'azione. Supponiamo di essere interessati a sapere quando e come una certa cosa accadrà. Con il nostro codice possiamo registrare un gestore di eventi verso l'azione che ci interessa. Ad ogni singola azione possono essere associati più gestori. Quando l'azione attesa è eseguita, tutti i gestori di eventi ad essa collegati vengono attivati, propagando l'evento. Un esempio pratico ce lo forniscono le interfacce grafiche: un evento esemplificativo è la pressione di un bottone presente nella finestra dell'applicazione. Il programmatore, in condizioni normali, è interessato a sapere quando l'utente clicca su uno dei bottoni mostrati. Per questo registra un gestore di eventi idoneo. Il risultato è che, alla pressione del bottone, uno o più metodi vengono automaticamente richiamati. Dentro questi metodi, naturalmente, va inserito il codice che deve fornire una risposta al comando dell'utente. Ad esempio, se il programma è una videoscrittura e sul bottone da premere c'è scritto “Salva”, è probabile che i metodi ad esso collegati mediante i gestori di eventi dovranno eseguire i passi necessari per salvare il documento corrente e soddisfare, così, la richiesta dell'utente. Gli eventi, in C#, sono una particolare forma di delegati. Gli eventi appaiono come membri di una classe, al pari delle proprietà e dei metodi. La sintassi è:

```
event nome-delegato nome-evento;
```

La clausola nome-delegato specifica il tipo di delegato associato all'evento, che deve essere stato definito in precedenza esternamente alla classe; nome-evento, invece, è semplicemente il nome che il programmatore sceglie di dare all'elemento. Passiamo ad un esempio concreto:

```
delegate void NotificaRisultato(int quanteSomme,
                                            int risultato);
class Somma {
  public event NotificaRisultato ogniCinqueSomme;
  private int totale = 0;
  private int sommeEffettuate = 0;
  public void aggiungi(int n) {
    totale += n;
    sommeEffettuate++;
    if (sommeEffettuate % 5 == 0) {
      ogniCinqueSomme(sommeEffettuate, totale);}}
}
class Test {
  static void stampaRisultato(int quanteSomme,
                                            int risultato) {
    System.Console.WriteLine("Sono stati introdotti "
                            + quanteSomme + " numeri");
    System.Console.WriteLine("Somma ottenuta: "
                                       + risultato); }
  public static void Main() {
    Somma s = new Somma();
    s.ogniCinqueSomme += new
                    NotificaRisultato(stampaRisultato);
    while (true) {
      System.Console.Write("Inserisci un numero intero: ");
      int n = System.Int32.Parse(
                          System.Console.ReadLine());
      s.aggiungi(n);}}
}
```

Se desideri contattare l'autore di questo articolo scrivi a carlo.pelliccia@ioprogrammo.it

CRONACA DI UN EVENTO

Cominciamo l'analisi osservando la classe *Somma*. Scopo di questo elemento è fornire oggetti che possano memorizzare una somma. Si comincia da zero, ovviamente, ed il metodo *aggiungi()* può essere sfruttato per incrementare il totale rappresentato dall'oggetto.
La classe definisce un evento:

```
public event NotificaRisultato ogniCinqueSomme;
```

Il delegato associato a questo evento è:

```
delegate void NotificaRisultato(int quanteSomme,
                                            int risultato);
```

Andiamo ad esaminare il metodo *aggiungi()*.
Il codice ricorda, attraverso l'intero *sommeEffettuate*, quante volte il metodo sia stato richiamato. Ogni cinque somme vengono attivati i gestori di evento associati a *ogniCinqueSomme:*

```
if (sommeEffettuate % 5 == 0)
{
  ogniCinqueSomme(sommeEffettuate, totale);
}
```

La propagazione di un evento, come è possibile osservare, si esegue richiamando il nome dell'evento come se questo fosse un metodo, fornendo inoltre gli eventuali argomenti richiesti.

Andiamo ora a vedere il codice della classe *Test*. Questa contiene un metodo, *stampaRisultato()*, conforme al delegato *NotificaRisultato*, e quindi adatto anche all'evento *ogniCinqueSomme* di *Somma*. Nel *Main()*, dopo aver realizzato un'istanza di *Somma*, il metodo *stampaRisultato()* viene registrato come gestore dell'evento *ogniCinqueSomme*. Tutto è molto semplice, giacché si ragiona come per i delegati:

```
Somma s = new Somma();
s.ogniCinqueSomme += new
                    NotificaRisultato(stampaRisultato);
```

Eseguendo il programma, verrà continuamente richiesto all'utente l'inserimento di un valore intero

Fig. 1: ***Il nostro piccolo esempio alle prese con la sbrie di fibonacci.***

(*CTRL + C* per interrompere il programma). Ogni cinque valori inseriti, il metodo *aggiungi()* di *Somma* attiverà l'evento, come spiegato prima. Di conseguenza saranno richiamati tutti i gestori ad esso collegati, nel nostro caso il solo *stampaRisultato()*. Sulla console apparirà in output un messaggio che informerà l'utente sul numero di somme effettuate e sul risultato ottenuto. Più gestori possono essere registrati verso un solo evento, sfruttando gli operatori + e += come per i delegati; e sempre come per i delegati, un gestore può essere rimosso dalla catena con – o con –=.

Carlo Pelliccia

Chiudiamo il corso e l'intero ciclo sull'ottimizzazione

Ottimizzazione del codice (parte terza)

In questa ultima puntata analizzeremo i concetti per migliorare le performance del codice. In particolare, focalizzeremo l'attenzione sui valori di ritorno delle funzioni.

Nel corso delle ultime due puntate abbiamo visto dapprima le ottimizzazioni preventive (quelle che possono precedere la scrittura del codice), quindi le ottimizzazioni inerenti la codifica (tutti gli accorgimenti che consentono di migliorare le prestazioni del software, e che possono essere seguiti durante la vera e propria scrittura del codice). Abbiamo dato delle linee guida che ricordiamo nuovamente:

- la velocità di un programma che non funziona è irrilevante;
- la fase di ottimizzazione deve essere posticipata il più possibile;
- il miglior compilatore è tra le nostre orecchie;
- si ottimizzano i programmi troppo ingombranti o troppo lenti;
- la rapidità di esecuzione non è legata alla lunghezza del codice.

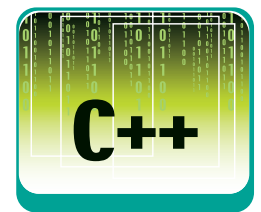
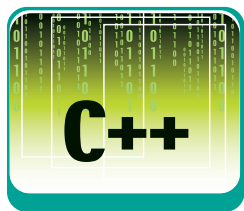
È tenendo sempre presente queste osservazioni che andiamo a guardare in questo appuntamento quelle che sono le ottimizzazioni finali del codice.

OTTIMIZZAZIONE DEI VALORI DI RITORNO

Quando scriviamo le nostre funzioni, dobbiamo sempre tenere a mente che il C++ crea e distrugge oggetti in punti del codice che non sono affatto ovvi.
Ad esempio, parlando di liste di inizializzazione, nella scorsa puntata abbiamo visto che quando viene chiamato un costruttore che non prevede tali liste, le variabili che rappresentano le proprietà dell'oggetto creato vengono comunque istanziate, in maniera implicita, prima dell'istruzione iniziale del corpo del costruttore.
Un qualcosa di analogo succede quando in una funzione restituiamo un risultato di un qualsiasi tipo, sia esso un intero o di un tipo da noi definito.
Esaminiamo un semplice esempio:

```
complex<double> CreaCpx(double Re,double Im) {
   complex<double> num_comp(Re,Im);
   return num_comp;
}
```

In questo esempio abbiamo fatto uso di uno dei tipi definiti nella STL, il tipo complex; esso si usa per rappresentare i numeri complessi, ed è definito mediante template per poter così usare i numeri complessi basati su più tipi possibili.
Nella STL sono anche presenti, per inciso, delle specializzazioni di questo tipo generico relativamente ai numeri complessi basati su tipi *float, double* e *long double*. In questa funzione così semplice, che si limita a costruire un numero complesso dati due numeri di tipo *double*, sono presenti in realtà moltissime inefficienze: ci possiamo infatti chiedere prima di tutto se sia proprio necessaria questa funzione, visto che a tutti gli effetti è un wrapper del costruttore dei numeri complessi, cioè una funzione contenitore il cui scopo è permettere di chiamare il costruttore dei numeri complessi in un altro modo, e le con-

siderazioni che ne scaturiscono fanno parte della fase di ottimizzazione preventiva.
A questo si può aggiungere una considerazione non banale: come già accennato esistono delle specializzazioni del tipo *complex* già pronte, reperibili nella STL, e una di queste è proprio l'implementazione del tipo *complex* basata sui *double;* quindi, perché non usare direttamente quella invece del tipo *complex<double>?*
Con ogni probabilità non saremmo in grado di scrivere un codice migliore e più prestante di quello che ci fornisce gratuitamente la STL. Anche questa è una considerazione che ha a che fare con l'ottimizzazione preventiva. Se poi guardiamo la lista dei parametri passati alla funzione ci accorgiamo che essi sono passati per valore: questo significa che si dovranno avere delle variabili temporanee per la copia dei parametri e, facendo le considerazioni già svolte parlando del passaggio dei parametri, giungeremmo alla considerazione che molto più prestante sarebbe stato passare tali parametri come riferimento costante.

APPROFONDIMENTI

A chi volesse approfondire la conoscenza delle librerie standard C++, consigliamo il validissimo libro "*Thinking in C++ - 2nd ed. - Volume 2*" di *Bruce Eckel* e *Chuck Allison*, che rappresenta sicuramente un ottimo riferimento per i programmatori più avanzati (o aspiranti tali) ed è oltretutto disponibile gratuitamente per il download, partendo dall'indirizzo http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html

Se volete invece dare un'occhiata a una reference delle funzioni standard del C per la manipolazione di stringhe (o meglio: di array di caratteri terminati da "\0" :-) consultate l'indirizzo: http://www.cplusplus.com/ref/cstring/

Online è inoltre disponibile l'utilissimo "C++ Annotations" all'URL: http://www.icce.rug.nl/documents/ **che merita di essere visto (e letto) almeno una volta.**

Facciamo finta però che tutte queste considerazioni non abbiano senso, e che la funzione sia concepibile solo in questa forma: anche in questo caso mancherebbe ancora una possibile ottimizzazione, quella del valore di ritorno.
La nostra funzione, infatti, all'atto della restituzione del valore dovrà creare un oggetto temporaneo di tipo *complex<double>* per contenere il risultato (una copia dell'oggetto *num_comp* creato al suo interno); in altre parole, la nostra funzione crea un oggetto temporaneo al suo interno, la variabile *num_comp,* usando come valori i parametri che riceve in ingresso, in effetti non restituisce tale oggetto ma solo il suo valore! Questo comporta che *num_comp* andrà distrutta al termine dell'esecuzione della funzione (e qui verrà quindi chiamato l'opportuno distruttore), ma soprattutto che il valore di *num_comp* dovrà essere copiato in un oggetto temporaneo che contiene il risultato della funzione (con conseguente spreco di spazio e di tempo, per via della chiamata al costruttore di copia). Tutte queste inefficienze potevano essere evitate mediante un semplice accorgimento, che possiamo osservare nella versione riveduta della funzione:

```
complex<double> CreaCpx(double Re, Double Im)
{
   return (complex<double>(Re,Im));
}
```

Interpretando alla lettera il corpo della funzione cogliamo subito una sottile, ma fondamentale, differenza: non restituiamo semplicemente un risultato, ma un vero e proprio oggetto senza nome. Questo accorgimento permette al compilatore di compiere una piccola magia: invece di copiare il risultato della funzione nella variabile/oggetto di destinazione, esso può costruire tale oggetto-risultato direttamente nello spazio di memoria dell'oggetto che conterrà tale risultato al di fuori della funzione, evitando così di effettuare copie e di chiamare, quindi, costruttori di copia. La magia è possibile perché il compilatore "capisce" che non avrà a che fare con valori temporanei, e potrà lavorare con gli oggetti del programma da cui la funzione è stata chiamata.
Se avessimo usato una variabile temporanea, il compilatore sarebbe stato costretto a rispettare il nostro volere, e avrebbe perciò dovuto costruire tale variabile, copiarla in un oggetto temporaneo contenente il risultato della funzione, e quindi distruggere tale variabile temporanea al termine dell'esecuzione della funzione.
Notiamo infine un dettaglio che però mette in risalto l'importanza della questione.
Le definizioni più recenti del C++ standard sono strutturate in modo che per un compilatore, sotto opportune ipotesi, sia possibile evitare questi oggetti temporanei contenenti i risultati delle funzioni. Questo comporta il fatto che, forse, tra qualche anno i compilatori ottimizzeranno in automatico i valori di ritorno delle funzioni, senza costringerci a ottimizzare "manualmente" il codice.

VALORI DI RITORNO E PARAMETRI PER RIFERIMENTO

Alla luce di quanto visto nel paragrafo precedente, sarebbe una agognata possibilità quella di poter fare una cosa di questo tipo:

```
complex<double>& CreaCpx(double Re,double Im)
{
   complex<double> num_comp(Re,Im);
   return num_comp;
}
```

In pratica restituire come risultato della funzione non una copia dell'oggetto ma l'oggetto stesso (tramite un riferimento ad esso), ferma restando la possibilità di costruire tale oggetto nel corpo funzione, con tutta la flessibilità semantica che questo consentirebbe. Questa semplice modifica comporterebbe il non dover fare null'altro per ottimizzare il nostro codice (sempre nelle ipotesi precedentemente poste che il meccanismo con cui opera la funzione e la lista di parametri siano necessariamente da lasciare immutati). Una sola pressione di tasti e avremmo fatto tutto quanto nelle nostre facoltà per ottenere il massimo. Magari fosse così semplice! In realtà l'aggiunta di quella *"&"* al tipo restituito è una cosa pericolosissima, in quanto non solo non è un errore a livello sintattico (nel senso che un compilatore potrebbe non rilevare alcuna anomalia in quanto scritto), ma potrebbe non essere considerato un errore neanche a livello semantico. Come abbiamo detto, l'oggetto *num_comp* verrà distrutto al termine della funzione, quindi un riferimento a *num_comp* punterà in realtà non al risultato ma a un'area di memoria dominata potenzialmente dal caos (non ci è dato sapere cosa succederà nell'area di memoria occupata da *num_comp* dopo che tale area verrà deallocata e quindi, potenzialmente, riutilizzata). Però questo per il compilatore non è un errore, e, nel migliore dei casi, probabilmente avremo al massimo un warning, cioè un avviso di una possibile fonte di errore (in questo caso, a run-time). Appurato che una soluzione come questa non è percorribile, diciamo anche che ne esiste un surrogato. Lo stratagemma consiste nel passare alla funzione un parametro che è in realtà il contenitore del risultato. Ad esempio nel nostro caso potremmo fare così:

```
void CreaCpx(double Re, double Im,
                                  complex<double>& res)
{
   complex<double> num_comp(Re,Im);
   res = num_comp;
}
```

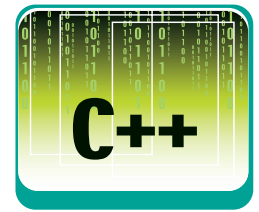
Si notino due cose: il tipo ritornato non è più *complex<double>* ma *void* (in realtà, potremmo far comunque ritornare qualcosa, ad esempio un bool che segnali la corretta esecuzione delle operazioni della funzione, ma comunque il risultato vero e proprio sarà contenuto in uno dei parametri), e il risultato sarà una delle variabili passate (per riferimento) come parametro. A questo proposito vogliamo inoltre far notare l'importanza dell'utilizzo della particella *"const"* quando si effettua il passaggio di un parametro che resterà immutato, per riferimento anziché per valore, per evitarne la copia. La *const* è quindi utilizzata, oltre che per evidenti motivi semantici, anche per distinguere quest'ultimo caso dal caso precedente di restituzione di un risultato tramite parametro passato per riferimento.

UN UTILIZZO ATTENTO DELLE FUNZIONI VIRTUALI

Sebbene siano uno dei grossi vantaggi dell'utilizzo di linguaggi potenti come il C++, le funzioni virtuali (cioè le funzioni contraddistinte dalla clausola *virtual*, per le quali l'invocazione è decisa a run-time) costituiscono un argomento "scottante" nel campo delle ottimizzazioni del codice. L'utilizzo di questo tipo di funzione infatti, comporta un overhead di prestazioni dovuto alla necessità di utilizzare un livello aggiuntivo di indirezione.

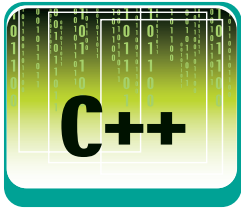
Una singola funzione virtuale in una classe richiede che ogni oggetto della classe, e ogni oggetto derivato, contengano un puntatore aggiuntivo; questo può avere un effetto negativo sulle dimensioni di oggetti piccoli che possono arrivare anche a raddoppiare.

Ad esempio la seguente classe

```
class OggettoVirtuale
{
   private:
      int valore;
   public:
      OggettoVirtuale () { valore = 0 }
      virtual ~OggettoVirtuale () {}
      virtual int funzione() const { return (valore); }
};
```

I seguenti libri sono stati usati come base per la puntata corrente:

• ZEN OF CODE OPTIMIZATION: THE ULTIMATE GUIDE TO WRITING SOFTWARE THAT PUSHES PCS TO THE LIMIT
Michael Abrash

• C++ OPTIMIZATION STRATEGIES AND TECHNIQUES
Pete Isensee
http://www.tantalon.com/pete/cppopt/main.htm
punto di riferimento per chiunque volesse approfondire l'argomento. Ne consigliamo vivamente la lettura.

• OPTIMIZING C++ - THE WWW VERSION
Steve Heller
ottimo libro consultabile online e stampabile.
http://www.steveheller.com/opt/.

genera degli oggetti di dimensione 8 byte (cioè sizeof(OggettoVirtuale) == 8). Una possibile conversione di questa classe in una struttura che non faccia uso di funzioni virtuali è la seguente:

```
class OggettoNONVirtuale
{  private:
        int valore;
   public:
        OggettoNONVirtuale () { valore = 0 }
        ~OggettoNONVirtuale () {}
        int funzione() const { return (valore); }
};
```

NOTA

NON TUTTI I WRAPPER VENGONO PER NUOCERE

Sebbene il piccolo wrapper inserito, a scopo didattico, in questa lezione sia sostanzialmente inutile, nei casi reali a volte capita di dover progettare dei moduli che consentano di raggruppare varie funzionalità di una classe, senza per questo dovere riscrivere l'intero codice che la compone. In casi in cui si abbia un programma molto grande e si debba modificare il comportamento di una classe, invece di sostituire tale classe in toto (e quindi andare a rintracciare tutti i tratti di codice in cui essa compare per sostituirli uno ad uno, cosa che per software estesi è improponibile) potremmo semplicemente creare un wrapper, cioè una classe "fantoccio" che fornisca all'esterno la nuova interfaccia ma che utilizzi al suo interno il codice preesistente, che in questo modo non andrà "buttato via".

La dimensione degli oggetti relativi è di 4 byte, esattamente la metà, per una classe che praticamente ha le stesse funzionalità. Oltre alle dimensioni maggiori, l'utilizzo di funzioni virtuali causa un appesantimento a tempo di esecuzione, in quanto bisogna inizializzare una VFT (virtual function table, cioè la struttura dati che si occupa della gestione delle funzioni virtuali) per ogni oggetto che si crea. Inoltre anche la chiamata stessa di una funzione virtuale è più lenta rispetto alla chiamata di una funzione non virtuale. Alla luce di quanto esposto è consigliabile non utilizzare (tenendo sempre a mente, le regole guida per una buona metodologia di ottimizzazione) le funzioni virtuali laddove queste non siano strettamente necessarie. C'è però da dire che, qualora si abbia davvero bisogno di utilizzare un meccanismo di questo tipo, difficilmente si riuscirà a svilupparne uno più efficiente di quello messo a disposizione dal C++. Si dovrebbe infatti inserire in ogni classe una variabile *(flag)* che ne indichi il tipo per il riconoscimento a runtime e, probabilmente, uno switch che controlli il valore di tale flag. Questo meccanismo è molto più soggetto a errori e meno leggibile dell'utilizzo delle clausole virtual, senza che ciò porti a un effettivo vantaggio in termini di prestazioni, in quanto la gestione "nativa" delle VFT è di gran lunga più performante. Alcuni compilatori consentono di impostare dei parametri relativi all'inizializzazione delle VFT in modo da rendere il codice più veloce. È il caso ad esempio della direttiva

```
__declspec(novtable)
```

che consente di disabilitare la creazione della VFT nel costruttore di un oggetto.
Questo normalmente produce un effetto deleterio, tranne nel caso di classi base astratte, per le quali non c'è nessuna necessità di inizializzare i puntatori della VFT, in quanto questi saranno certamente creati nella fase di istanziazione degli oggetti delle classi derivate (ricordiamo che non è possibile creare oggetti di classi contenenti funzioni virtuali pure). La direttiva appena citata è però specifica per i compilatori Microsoft (non fa quindi parte dello standard del C++). Un utilizzo di tale caratteristica implica, quindi, una ricerca da parte del programmatore della funzionalità corrispondente (se presente) all'interno delle opzioni messe a disposizione dallo specifico compilatore utilizzato.

Se hai suggerimenti, critiche, dubbi o perplessità sugli argomenti trattati e vuoi proporle agli autori puoi scrivere agli indirizzi:

alfredo.marroccelli@ioprogrammo.it e marco.delgobbo@ioprogrammo.it

Questo contribuirà sicuramente a migliorare il lavoro di stesura delle prossime puntate.

CONCLUSIONI

Abbiamo terminato, con questa puntata, l'argomento delle ottimizzazioni del codice C++, nonché il ciclo di lezioni relative a questo corso. Abbiamo cercato, nel corso di queste 25 puntate, di coprire il più vasto numero di argomenti del linguaggio, partendo dalle basi sino ad arrivare ad argomenti abbastanza complessi e delicati come ad esempio quello delle ultime lezioni. Il nostro scopo è sempre stato quello di inserire concetti teorici fondamentali all'interno di implementazioni pratiche in modo da rendere subito comprensibile il concetto che si stava esponendo. Speriamo di essere riusciti nel nostro intento di realizzare un corso completo, esauriente e *"user friendly"* e desideriamo ringraziare tutti i lettori che ci hanno seguito in questo viaggio, e in particolar modo quelli che (numerosi) ci hanno contattato tramite posta elettronica per esprimere opinioni, dubbi e suggerimenti.
Un grazie di cuore e... a presto!

Alfredo Marroccelli
Marco Del Gobbo

Impara ad usare il polimorfismo con Java

La classe che cambiò forma

Come per magia, può un oggetto assumere diverse forme? Eccome, se può! Il polimorfismo è una delle caratteristiche più importanti di Java e degli altri linguaggi orientati agli oggetti.

Il mese scorso hai imparato cos'è l'ereditarietà. Questo mese continueremo il discorso per parlare di quella che è forse la caratteristica più importante della programmazione orientata agli oggetti: il polimorfismo. Questa lezione può sembrare semplice, perché non incontrerai nessuna nuova parola chiave. Ma i concetti nei quali stai per imbatterti hanno ramificazioni profonde. Non prendere sottogamba il polimorfismo: esistono programmatori esperti che non hanno mai imparato ad usarlo come si deve. Se riuscirai a capirlo e a sfruttarlo a fondo, ti assicuro che non guarderai indietro.

OBIETTIVI

1) Imparerai cos'è l'upcasting.
2) Scriverai le tue prime operazioni polimorfiche.
3) Vedrai come si può usare il polimorfismo per costruire programmi estensibili.

Bando agli indugi e...

...COMINCIAMO!

Guarda questa classe:

```
class Persona {
  public void saluta() {
    System.out.println("Salve!"); }
  public void daiIndicazioniPer(String destinazione) {
    System.out.println("Per andare in " + destinazione
              + " si deve girare in fondo a destra."); }
  public void riceviInsulto() {
    System.out.println("Ma come ti permetti?"); }
  public void medita() {
    System.out.println("mumble, mumble..."); }
}
```

Puoi creare una *Persona* e imbastire una semplice conversazione:

```
Persona passante = new Persona();
passante.saluta();
passante.daiIndicazioniPer(“Piazza della Vittoria”);
```

La *Persona* risponde alla conversazione stampando sullo schermo:

```
Salve!
Per andare in Piazza della Vittoria si deve girare in
fondo a destra.
```

Ecco ora una classe che eredita da *Persona* e ridefinisce alcuni metodi (cioè ne fa l'override):

```
class Vecchietto extends Persona {
  public void saluta() {
    System.out.println("I miei rispetti alla Sua signora."); }
  public void riceviInsulto() {
    System.out.println("Lei non sa chi sono io!"); }
  public void passeggia() {
    System.out.println("Tu-de-dum..."); }
}
```

Se guardi la rappresentazione in UML di queste due classi vedrai che il *Vecchietto* riceve in ereditarietà tutti i metodi della *Persona,* ma ne ridefinisce due a modo proprio e ne aggiunge uno nuovo di zecca. Quindi *Vecchietto* eredita solo in parte il codice di *Persona,* ma ne eredita completamente le caratteristiche – per essere più precisi, la sua interfaccia. L'interfaccia di una classe è l'insieme dei suoi metodi e dei suoi campi non privati. I metodi *saluta()* e *daiIndicazioniPer()* sono nell'interfaccia di *Persona,* quindi tutte le classi che ereditano da *Persona* devono avere questi metodi. Una sottoclasse non può “eliminare” i metodi della sua superclasse.
Di solito si dice che una classe è legata alla sua su-

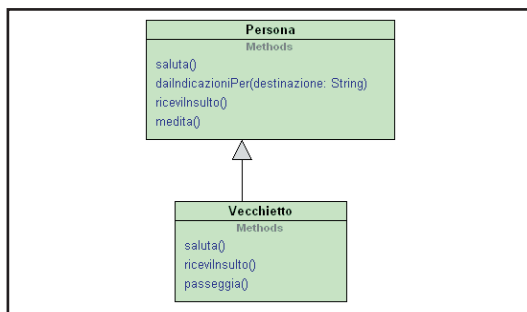



Fig. 1: **UML: la relazione "is_a" tra vecchietto e persona.**

perclasse da una relazione *"is_a"* (per dirla in italiano, da una relazione *"è un"*). Un *Vecchietto "è una" Persona,* cioè qualsiasi oggetto di tipo *Vecchietto* è anche un oggetto di tipo *Persona.* Quindi puoi scrivere una riga come questa:

```
Persona p = new Vecchietto();
```

Come forse ricordi, l'operazione di *new* restituisce un reference all'oggetto appena creato. Di solito, per evitare che questo reference e il relativo oggetto vadano persi, devi assegnare il reference ad una variabile. È quello che ho fatto anche in questo caso, con una importante differenza: la variabile non ha esattamente lo stesso tipo dell'oggetto che ho appena creato. L'oggetto è un *Vecchietto,* mentre la variabile *p* è fatta per contenere un reference di tipo *Persona.* Allora, come mai il compilatore Java accetta questo codice? Perché ogni *Vecchietto* è anche una *Persona* e quindi posso trattarlo come tale.
Per essere precisi, l'oggetto non viene convertito. Quello che cambia è solo il reference. Il nostro oggetto è ancora un *Vecchietto,* ma il codice ha deciso di "dimenticarsi" di avere a che fare con un *Vecchietto* e di trattarlo come una più generica *Persona.* Ora puoi chiamare tutti i metodi di *Persona* sulla variabile *p,* ma non puoi più chiamare i metodi di *Vecchietto.* Nei casi come questo, nei quali assegni un oggetto di una sottoclasse ad un reference ad una superclasse, si dice che fai un upcasting, cioè una *"conversione di tipo verso l'alto".* L'upcasting deve il suo nome al fatto che si tratta di un *"cast"* (una conversione di tipo) e che nei diagrammi si disegnano di solito le superclassi in alto e le sottoclassi in basso. Ora sai cos'è l'upcasting, ma non sai ancora a cosa serve. Per quale bizzarro motivo dovremmo "dimenticarci" il tipo di un oggetto? Perché in questo modo possiamo scrivere del codice che conversa con le persone, e questo codice potrà automaticamente conversare anche con i vecchietti.

```
private void conversa(Persona p) {
  p.saluta();
  p.daiIndicazioniPer("Vicolo Stretto"); }
```

Questo metodo prende una *Persona* e si intrattiene in una brevissima conversazione. Cosa succede se passiamo un *Vecchietto* a *conversa()*? Succede la stessa cosa che succede nella riga di codice che hai visto poco fa: l'oggetto *Vecchietto* viene "convertito verso l'alto" al tipo *Persona.* Prima l'upcasting avveniva durante l'assegnamento, ora avviene durante il passaggio del parametro. Il metodo *conversa()* non è obbligato a distinguere tra *Vecchietti* e *Persone.* In effetti il metodo non sa nemmeno che esista un tipo *Vecchietto:* conosce solo le *Persone.* Dato che la classe *Persona* definisce un metodo *saluta()* e un metodo *daiIndicazioniPer(),* il metodo *conversa()* può chiamare questi metodi. Se *conversa()* cercasse di chiamare un metodo specifico del *Vecchietto,* come *passeggia(),* il compilatore di Java si arrabbierebbe – e farebbe bene, visto che il metodo potrebbe ricevere una *Persona* che non è un *Vecchietto,* e quindi non può passeggiare. Ho blaterato abbastanza - è ora di passare agli esperimenti concreti. Cosa succede se passi un *Vecchietto* al metodo *conversa()*?

BINDING DINAMICO

Il polimorfismo si basa su un meccanismo che si chiama binding dinamico. In un linguaggio tradizionale come il C, tutte le chiamate a funzione vengono risolte durante la compilazione. Questo significa che il programma contiene sono "salti" diretti da una parte all'altra del codice. Nei linguaggi orientati agli oggetti, invece, le chiamate a metodo sono risolte solo mentre il programma sta girando. Per questo motivo un programma può chiamare metodi diversi a seconda dell'oggetto che gli viene passato.

UN TRUCCO MENTALE DI STILE JEDI

```
class Conversazioni {
  private static final String DESTINAZIONE = "Via Larga";
    public static void main(String[] args) {
    System.out.println("--- Conversazione 1: Persona");
    Persona gino = new Persona();
    conversa(gino);
    System.out.println("--- Conversazione 2: Vecchietto");
    Vecchietto piero = new Vecchietto();
    conversa(piero); }
  private static void conversa(Persona p) {
    System.out.println("> Buongiorno!");
    p.saluta();
    System.out.println("> Sto cercando " +
                                    DESTINAZIONE + "...");
    p.daiIndicazioniPer(DESTINAZIONE);
    System.out.println("> Ma vai a quel paese!");
    p.riceviInsulto(); } }
```

Ho reso *static* il metodo *conversa(),* perché altrimenti non lo avrei potuto chiamare direttamente dal *main().* L'ho anche reso *private,* perché non viene mai chiamato dall'esterno della classe ed è sempre buona regola rendere le cose quanto meno visibili possibile. Infine ho definito una costante *DESTINAZIONE* per evitare di scrivere la stessa stringa più volte (è sempre meglio usare le costanti in questi casi – sono più esplicite, e se fai un errore di battitura mentre usi la costante il compilatore se ne accorge subito). A questo punto ti potrebbe sorgere un dubbio. Esistono due versioni dei metodi *saluta()* e *riceviInsulto().* Ma allora, quali versioni dei due metodi vengono eseguite quando conversi con il *Vecchietto?* Come dicono gli inglesi, "la prova del budino è nel

mangiarlo" - quindi fai girare il codice e guarda cosa salta fuori.

```
--- CONVERSAZIONE 1: PERSONA
> Buongiorno!
Salve!
> Sto cercando Via Larga...
Per andare in Via Larga si deve girare in fondo a destra.
> Ma vai a quel paese!
Ma come ti permetti?

--- CONVERSAZIONE 2: VECCHIETTO
> Buongiorno!
I miei rispetti alla Sua signora.
> Sto cercando Via Larga...
Per andare in Via Larga si deve girare in fondo a destra.
> Ma vai a quel paese!
Lei non sa chi sono io!
```

Questo risultato potrebbe stupirti molto... o forse per niente. Se sei già un programmatore ma non hai esperienza di object-oriented, sarai probabilmente rimasto colpito. Anche se *conversa()* fa riferimento solo alla classe *Persona,* il codice che viene chiamato non è necessariamente quello di *Persona,* ma quello del particolare oggetto che gli viene passato. Nel caso del *Vecchietto,* il codice di *daiIndicazioniPer()* è lo stesso di *Persona,* ma *saluta()* e *riceviInsulto()* sono cambiati. Quindi il metodo *conversa()* si comporta in modo diverso nei due casi.
Questo è un esempio di polimorfismo. *Polimorfico,* in greco, significa "che assume diverse forme". I metodi *saluta()* e *riceviInsulto()* sono polimorfici, cioè assumono forme diverse nelle varie classi imparentate. Di riflesso il metodo *conversa()* è polimorfico, perché si comporta in modo diverso a seconda dell'oggetto che gli viene passato.

SISTEMI CHE CRESCONO

La cosa più importante del polimorfismo è che ti permette di scrivere programmi estensibili. Grazie al polimorfismo, un client è interessato a quello che gli oggetti fanno, ma non a *come* lo fanno. Con un po' di esperienza potrai sfruttare questa "ignoranza" per costruire programmi nei quali alcune funzionalità cambiano senza che il resto del programma debba essere modificato. Questo aspetto del polimorfismo funziona così bene che puoi inventare nuove sottoclassi di *Persona* anche dopo aver scritto il metodo *conversa()* e passarli al metodo. Tutto continuerà a funzionare tranquillamente, e *conversa()* resterà ignaro del fatto che i suoi parametri stanno facendo qualcosa di nuovo.
Ecco la mia personale soluzione dell'*Esercizio 1*:

```
class Coatto extends Persona {
  public void saluta() {
    System.out.println("Ciao tipo!"); }
  public void riceviInsulto() {
    System.out.println("Aho', vie' qua che te rompo!"); }
}
```

Se passi un *Coatto* a *conversa()* otterrai:

```
> Buongiorno!
Ciao tipo!
> Sto cercando Via Larga...
Per andare in Via Larga si deve girare in fondo a destra.
> Ma vai a quel paese!
Aho', vie' qua che te rompo!
```

Nulla ci impedisce di estendere la gerarchia delle *Persone* su più livelli, cioè di ereditare da una classe che a sua volta eredita da un'altra classe:

```
class VecchiettoSordo extends Vecchietto {
  public void daiIndicazioniPer(String destinazione) {
    System.out.println("Può ripetere, scusi?"); }
  public void riceviInsulto() {
    System.out.println("Eh?");} }
```

Ecco il risultato della conversazione con un *VecchiettoSordo:*

```
> Buongiorno!
I miei rispetti alla Sua signora.
> Sto cercando Via Larga...
Puo' ripetere, scusi?
> Ma vai a quel paese!
Eh?
```

In Fig. 2 puoi vedere la gerarchia delle *Persone.*

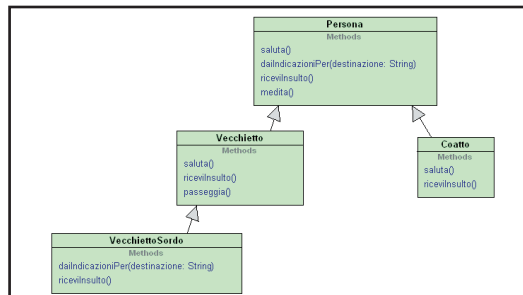



***Fig. 2:** Il nuovo diagramma delle classi illustra bene i più livelli della gerarchia.*

VecchiettoSordo ridefinisce i metodi *daiIndicazioniPer()* e *riceviInsulto()*, ed eredita il resto dal suo più vicino antenato. Quindi riceve *saluta()* e *passeggia()* da *Vecchietto,* e *medita()* da *Persona.* Puoi fare l'"upcast" di un *VecchiettoSordo* a *Vecchietto* oppure a *Persona.*
Nel primo caso puoi usare ancora tutti i metodi dell'oggetto, perché *VecchiettoSordo* non aggiunge metodi all'interfaccia di *Vecchietto;* nel secondo caso il metodo *passeggia()* diventa inaccessibile.

SEPARARE CIÒ CHE È UNITO

Ecco un esempio di come il polimorfismo può aiu-

Scrivi una nuova classe che eredita da *Persona*. Ridefinisci solo alcuni metodi di *Persona*, a tuo piacimento. Crea un oggetto della classe e passalo al metodo *conversa()*.

GIOVANE E DINAMICO

In alcuni linguaggi orientati agli oggetti, come il C++, si deve dire esplicitamente al compilatore quali metodi sono polimorfici e quali no. Se dichiari che un metodo è potenzialmente polimorfico (cioè che una sottoclasse può cambiarne il codice), allora il compilatore lo tratta in modo diverso dai normali metodi – cioè usa il binding dinamico. In Java questa distinzione non esiste: tutti i binding sono dinamici, quindi tutti i metodi sono potenzialmente polimorfici.

NOTA

ANIMALI COSTOSI

Nella gerarchia della classe *Animale* ho definito un metodo *specie()* che restituisce il nome dell'animale. Ma subito è sorto un problema: è chiaro quello che il metodo deve restituire nelle sottoclassi (*"cane"* per Cane, eccetera), ma non so bene cosa restituire nell'implementazione della superclasse Animale. Qual è la specie di un generico Animale? Avrai un problema simile quando risolverai l'Esercizio 2 e dovrai scrivere il metodo *Animale.prezzo()*.

ESERCIZIO 2

Implementa il metodo *prezzo()* nella gerarchia Animale. Fai girare il programma, e verifica che il risultato sia corretto.

ESERCIZIO 3

Aggiungi nuovi Animali alla gerarchia, e passali attraverso il metodo *accredita()*. Verifica che il metodo funzioni con qualsiasi Animale. Se porti il programma fuori dal *main()* puoi addirittura aggiungere nuovi Animali e passarli al metodo senza ricompilare la classe *Cassa3*.

tarti a "disaccoppiare" le diverse parti del tuo programma per renderlo più flessibile.
Guarda questo codice:

```
// la cassa di un negozio di animali
class Cassa1 {
  private long _totale;
  public void accredita(String animale) {
    if(animale.equals("pappagallo"))
      _totale += 10;
    else if(animale.equals("cane"))
      _totale += 5;
    else if(animale.equals("muflone"))
      _totale += 25;}
  public static void main(String[] args) {
    Cassa1 cassa = new Cassa1();
    cassa.accredita("pappagallo");
    cassa.accredita("cane");
    cassa.accredita("cane");
    System.out.println("TOT: " + cassa._totale); }
}
```

Ho usato l'operatore *'+='* per incrementare il campo privato *_totale*. Nota che *Cassa1* include il proprio *main()* di test, che fa la parte del client. Immagina che il *main()* sia un programma complicato che usa la classe *Cassa1*. Nota anche che il *main()* può accedere al campo *totale* anche se è privato, perché fa parte della stessa classe. Per ottenere il totale dall'esterno potresti scrivere un metodo *totale()* che restituisce il valore del campo senza esporre il campo stesso ai client. Cosa ne pensi di questo programma? Sicuramente funziona (e stampa come risultato *20*), ma personalmente lo trovo decisamente bruttino. Nonostante le sue minuscole dimensioni, il programma è poco estensibile. Se decidi di aggiungere un animale allora devi intervenire in due punti: devi creare una nuova classe che eredita da *Animale*, e devi anche modificare il metodo *accredita()* perché riconosca il nuovo animale e lo aggiunga al conto. Questo è un modo quasi garantito per introdurre dei bug nel codice: quando i metodi che usano gli animali sono centinaia, e tutti contengono catene di istruzioni condizionali come quelle di *accredita()*, è facilissimo dimenticarsi di modificare uno dei metodi di quando si introducono nuovi elementi nel programma, o sbagliare nello scrivere una delle stringhe (che appaiono tutte in almeno due punti). Come primo passo vorrei trasformare il parametro di *accredita()* in una gerarchia di oggetti. Ecco una seconda versione del programma, che sfrutta il polimorfismo:

```
class Animale {
  public String specie() {
    return "";} }
class Cane extends Animale {
  public String specie() {
    return "cane"; } }
class Pappagallo extends Animale {
  public String specie() {
    return "pappagallo"; } }
class Muflone extends Animale {
  public String specie() {
    return "muflone"; } }
class Cassa2 {
  private long _totale;
  public void accredita(Animale acquisto) {
    if(acquisto.specie().equals("pappagallo"))
      _totale += 10;
    else if(acquisto.specie().equals("cane"))
      _totale += 5;
    else if(acquisto.specie().equals("muflone"))
      _totale += 25; }
  public static void main(String[] args) {
    Cassa2 cassa = new Cassa2();
    cassa.accredita(new Pappagallo());
    cassa.accredita(new Cane());
    cassa.accredita(new Cane());
    System.out.println("TOT: " + cassa._totale);}
}
```

Ora i nomi degli animali sono stati trasferiti nella gerarchia di oggetti attraverso il metodo polimorfico *specie()*. I nostri problemi di duplicazione non sono ancora risolti. Però ora abbiamo una struttura ad oggetti, e possiamo eliminare quelle brutte istruzioni *if* a colpi di polimorfismo. La classe *Cassa2* deve continuamente mettere le mani negli oggetti della gerarchia *Animale*, estrarne dei dati, e in base a questi dati prendere delle decisioni. Pare che la complessità del nostro programmino sia il sintomo di un problema di design: la classe *Cassa2* si sta assumendo delle responsabilità che dovrebbero appartenere direttamente agli *Animali*. In particolare, fa tutto questo per calcolare il prezzo dei singoli animali. Allora, perché non creare un metodo *prezzo()* direttamente nella classe *Animale?* Così ciascun animale potrà calcolare autonomamente il proprio prezzo, e la cassa deve solo leggerlo e aggiungerlo al totale:

```
class Cassa3 {
  private long _totale;
  public void accredita(Animale acquisto) {
    _totale += acquisto.prezzo(); }
  public static void main(String[] args) {
    // (uguale a quello di Cassa2) }
}
```

Ora sì, che mi piace!
Spero che tu sia soddisfatto di quello che hai imparato questo mese. Ma non dormiremo sugli allori: il mese venturo ti aspettano altri aspetti dell'ereditarietà e del polimorfismo - e questa volta farai una scorpacciata di nuove parole chiave. A presto!

Paolo Perrotta

Impariamo a proteggerci dagli hacker

La crittografia con CAPICOM

Utilizzando un controllo Activex per la CryptoAPI, descriviamo come incorporare la firma digitale, o elettronica, e la crittografia nelle applicazioni Visual Basic.

Il termine PKI *(Public Key Infrastructure)*, viene utilizzato per descrivere tutto ciò che riguarda l'autenticazione degli utenti e la sicurezza sulla rete. PKI, dunque, ingloba i software di crittografia e quelli che amministrano i certificati digitali e le chiavi pubbliche e private. In altre parole il PKI si preoccupa di proteggere dati sensibili e garantisce sui componenti software utilizzati. I servizi di sicurezza basati sulla crittografia, vengono divisi in quattro rami: *crittografia a chiave privata* (crittografia simmetrica), *crittografia a chiave pubblica* (crittografia asimmetrica), *firma di crittografia* e *hash di crittografia.* In Windows i componenti che permettono agli sviluppatori di utilizzare i servizi di sicurezza crittografica sono: CryptoAPI e Cryptographic Service Providers (CSP). In Visual Basic queste funzionalità possono essere incorporate utilizzando CAPICOM (cioè CryptoAPI COM). In questo articolo descriveremo come implementare un'applicazione (Client) che utilizza i servizi di crittografia di Windows, in particolare descriveremo come crittografare/decrittare un testo (file o stringa), come utilizzare i certificati digitali, per "firmare" (garantire) il contenuto di un file, e vari concetti, sull'information security, che orbitano intorno a questi argomenti.

CryptoAPI e CSP

La CryptoAPI è un insieme di API *(Application Programming Interfacce)* che consentono di inserire, nelle applicazioni Win32-based, le funzionalità per la messa in sicurezza del software e delle trasmissione dati. In particolare permettono di utilizzare i servizi di autenticazione, di crittografia, di hash, di codifica e decodifica e di enveloping messages (imbustare dei documenti). La CryptoAPI aderisce allo standard PKCS #7 della RSA per quanto riguarda l'envelopment message, consente di codificare e decodificare i dati con ASN.1 (Abstract Syntax Notation One) e di amministrare certificati strutturati secondo lo standard X.509. La CryptoAPI con una particolare tecnica permette di verificare automaticamente i certificati avvalendosi di una lista di certificati attendibili. I CSP (Cryptographic Service Providers), invece, sono i provider dei servizi di crittografia.
I CSP implementano i servizi della CryptoAPI, forniscono algoritmi di crittografia più robusti.

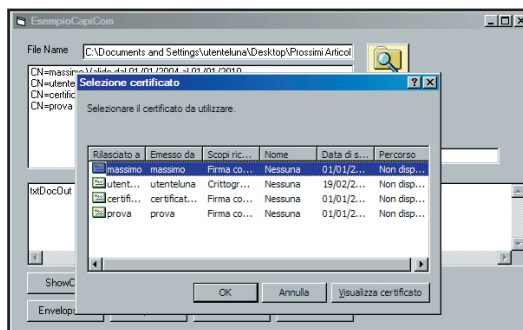
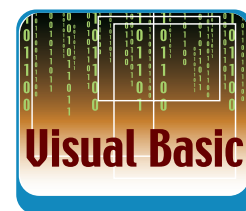
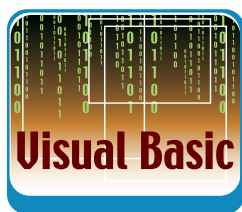



Fig. 1: *In figura è mostrato il form dell'applicazione.*

La crittografia con CAPICOM

CAPICOM è un Activex che fornisce un'interfaccia COM alla CryptoAPI, quindi consente di estendere i servizi di sicurezza di Windows a Visual Basic (VbScript, ASP, C++, …). CAPICOM non aggiunge altre funzionalità alle CryptoAPI ma le utilizza anche se con alcune limitazioni. Per esempio anche se CryptoAPI supporta vari algoritmi di hash, CAPICOM, per le firme, può utilizzare solo SHA-1. Inoltre c'è da specificare che i dati crittografati con CAPICOM possono essere letti solo con un'altra applicazione implementata con CAPICOM. Sottigliezze a parte CAPICOM può essere utilizzato per amministrare i certificati e le firme digitali, per l'envelop dei dati, per crittografare e decrittare e per varie altre cose.

CERTIFICATO DIGITALE

Come accennato i certificati digitali (la "firma digitale") servono per rendere sicuri i componenti Software e, in generale, le comunicazioni su Internet. Un certificato digitale garantisce che un componente Software proviene da una fonte attendibile, certificata da un'autorità di certificazione (CA) riconosciuta. In parole povere il certificato elettronico (o digitale) è una sorta di carta d'identità per Internet e la CA è "l'ufficio anagrafico". Un certificato digitale di solito contiene le seguenti informazioni: la chiave pubblica e le generalità del possessore cioè nome e cognome (se il possessore è una persona fisica) l'indirizzo e il nome della compagnia (quando invece si tratta di un server web); la data di scadenza della chiave pubblica (come nel caso della carta d'identità c'è una scadenza!) e il nome e la firma digitale della autorità di certificazione. In seguito vedremo che la chiave pubblica, e quella privata, servono per crittografare e decrittare un dato (messaggio, file ecc.). Sintetizzando, chi invia un messaggio lo cifra con la chiave pubblica del destinatario e quest'ultimo può leggerlo solo decodificandolo con la sua chiave privata (che in ogni caso resta segreta). Internet Explorer permette di controllare la lista dei certificati, e la lista dei CA di fiducia *(Trusted CAs)* presenti sul computer. Questo può essere fatto utilizzando il pulsante "Certificati" presente sulla scheda contenuto *(Content Tab)* della maschera Opzioni Internet. Internet Explorer, per proteggere il computer, prima del Download di un programma ne verifica l'identità attraverso la tecnologia Authenticode, cioè verifica se il programma, che si vuole scaricare, abbia un certificato elettronico valido che sia stato rilasciato da un'autorità riconosciuta e non sia ancora scaduto. Una cosa analoga succede quando ci si collega ad un sito protetto (quelli che iniziano con *https* invece che con *http*); il sito invia al navigatore il suo certificato in modo da essere riconosciuto ed assicurare che i dati sensibili, che saranno forniti, non verranno letti e modificati da altri. Dunque un certificato digitale, personale o di un sito, associa una chiave pubblica e una privata ad un'identità. La chiave privata, conosciuta solo dal proprietario, consente di porre una "firma digitale" o decrittare dei dati che qualcun altro ha crittografato utilizzando la sua chiave pubblica (nota a chiunque). Naturalmente tra chiave pubblica e privata c'è un legame deducibile attraverso un algoritmo matematico! Per gli esempi che presentiamo in questo articolo, potete creare dei certificati temporanei (validi solo sul vostro computer) attraverso i tool *SelfCert.EXE* o *MakeCert.exe*, che trovate nel sistema operativo o che potete scaricare dal sito della Microsoft.

Fig. 2: Un esempio di certificato generato con SelfCert.EXE di Office.

NOTA

SSL

SSL è il più comune protocollo (client /server) di sicurezza adoperato sulla rete; esso, per proteggere i dati, utilizza una combinazione di certificati digitali (chiavi pubbliche/ private) e crittografia (a 128 bit). I certificati ammessi possono essere sviluppati secondo vari standard, tra cui lo standard X.509.

Fig. 2: La finestra di Internet Explorer che permette di gestire i certificati.

INSTALLAZIONE DI CAPICOM

L'ultima release di CAPICOM è la 2.0.0.3 che può essere scaricata dal sito Microsoft. Essa è contenuta nel pacchetto *CAPICOM.EXE*. Questo contiene vari file (per esempio *CAPICOM.CAB*) e alcuni esempi pratici di utilizzo per i vari linguaggi supportati.
Per installare CAPICOM sul vostro computer, bisogna estrarre *CAPICOM.DLL*, dal file *CAPICOM .CAB* (che si trova nella directory */…/CAPICOM /x86/*), e poi registrarla, utilizzando *regsvr32 .exe*, con la seguente:

```
regsvr32.exe CAPICOM.DLL
```

Naturalmente prima bisogna individuare la directory in cui è contenuto il file (che potrebbe essere */…/CAPICOM/x86/*). Nella *Tabella 1* abbiamo riassunto le categorie in cui sono raggruppati gli oggetti, le interfacce ed i tipi che si possono gestire con *CAPICOM*. Di seguito descriveremo come utilizzare *CAPICOM* e introdurremo un'applicazione che consente di crittografare/decrittare una stringa (o un file), di mostrare i certificati elettronici presenti sul computer, di firmare il contenuto di un file e di imbustare un messaggio da trasmettere.

USO DI CAPICOM

Create un nuovo progetto EXE e referenziate la libreria *CAPICOM.DLL*, sulla form inserire degli oggetti che vi permettono di ricercare un file (un textbox, un Common Dialog ed un pulsante), una listbox (che utilizzeremo per mostrare i certificati digitali) e diversi pulsanti per abilitare i vari esempi che presenteremo. Prima di implementare gli esempi bisogna capire come sono organizzati i certificati registrati nel nostro computer. I certificati sono raggruppati in Store (quindi sono amministrabili attraverso gli oggetti della categoria *Certificate Store*) permanenti o temporanei, questi ultimi (presenti solo in memoria centrale) sono necessari, quando non si vogliono archiviare alcuni certificati negli store permanenti. Di solito in Windows sono definiti 3 store: MY (contenente i certificati dell'utente), Root e CA.
Ora possiamo descrivere il codice che mostra in una listbox i certificati di MY store.

COME MOSTRARE I CERTIFICATI

Nella procedura per mostrare i certificati utilizzeremo i seguenti oggetti.

1. Store che fornisce i metodi per scegliere ed amministrare i certificati negli Store. Esso tra l'altro presenta la proprietà *Certifications* che permette di selezionare la collezione di Certificate nello Store.

2. Certificate che rappresenta un singolo certificato digitale, permette di caricare il certificato da un file, determinare la validità del certificato, ecc.

La procedura per mostrare i certificati è la seguente (nel CD allegato alla rivista troverete un esempio più semplice).

```
Private Sub VisualizzaCerti_Click()
 Dim objStore As New CAPICOM.Store
 Dim objCertificate As CAPICOM.Certificate
 List1.Clear
 objStore.Open
 For Each objCertificate In objStore.Certificates
  List1.AddItem objCertificate.SubjectName & _
  " Valido dal " + CStr(objCertificate.ValidFromDate) & _
                   " al " + CStr(objCertificate.ValidToDate)
 Next
End Sub
```

In essa per aprire uno Store utilizziamo il metodo *Open* che ha tre parametri opzionali.
La sintassi di *Open* è

```
Open (StoreLocation, StoreName, OpenMode)
```

Il parametro *StoreLocation* indica il tipo e la location dello Store (memoria centrale, store dell'utente corrente, local-machine, Active Directory ecc.), il valore che utilizziamo nella procedura è quello di default cioè *CURRENT_USER_STORE*. *StoreName*, invece, è il nome dello Store che deve essere aperto, il suo valore di default è *My Store (CAPICOM_MY_STORE)*. Infine *OpenMode* indica in che modo lo Store deve essere aperta, il suo valore di default è *CAPICOM_STORE_OPEN_EXISTING_ONLY* (apri lo store solo in lettura). Dell'oggetto *Certification* invece utilizziamo le proprietà: *SubjectName* che fornisce il nome del titolare del certificato; *ValidFromDate* la data d'inizio validità e *ValidToDate* data di fine validità.

FIRMARE UN FILE

Con il successivo esempio vedremo come "firmare" un file. Gli elementi di CAPICOM che utilizzeremo sono:

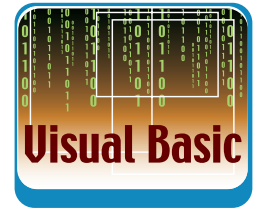
- **SignedData**, l'oggetto che permette di stabilire il contenuto (*content*) che deve essere firmato o verificato dopo la firma digitale;
- **Signer**, l'oggetto che permette di stabilire il firmatario (*Signer*) di un *SignedData* e che tra l'altro fornisce la collezione *AuthenticatedAttributes* di attributi di certificazione;
- **Attribute**, questo oggetto fa parte degli oggetti ausiliari e serve per fissare un singolo attributo di autenticazione.

Categoria	Descrizione
Certificate Store	L'insieme degli oggetti che permettono di amministrare i certificati archiviati (negli Store)
Digital Signature	Oggetti usati per amministrare la firma digitale dei dati
Enveloped Data	Oggetti che possono essere usati per l'enveloped data messages ("imbustare i dati") e per il decrypt dei dati "imbustati"
Data Encryption	Gli oggetti per crittografare/decrittare i dati
Oggetti accessori	Oggetti che permettono di svolgere diverse attività accessorie come: cambiare le caratteristiche di default, amministrare gli attributi dei certificati ecc.
Interoperability Interfaces	Interfacce che consentono ai derivati di CryptoAPI di lavorare insieme a CAPICOM
Enumeration Types	Tipi enumerativi utilizzati con CAPICOM

TABELLA 1 - I principali elementi di CAPICOM.

Prima di presentare la procedura facciamo notare che in essa non abbiamo incluso il codice che, attraverso un Common Dialog, permette di selezionare il file da firmare inserendo il suo nome e il path nel textbox *txtfilename*. Inoltre, notate che per selezionare un certificato utilizziamo la procedura *ScegliCertificato* e che il file firmato è salvato con il nome *"firmato"+"nomefileoriginario"*.

```
Private Sub firma_Click()
 On Error GoTo errore
 Dim cont As String
 Dim sig As String
 Dim Signedobj As New SignedData
 Dim Signer As New Signer
 Dim TempoSign As New CAPICOM.Attribute
 If ScegliCertificato(Signer) Then
 Else
   MsgBox "selezionare un certificato"
   Exit Sub
 End If
 Open txtfilename For Input As #1
 Input #1, cont
 Close #1
 Signedobj.Content = cont
 TempoSign.Name = CAPICOM_AUTHENTICATED_
                                   ATTRIBUTE_SIGNING_TIME
 TempoSign.Value = Now
 Signer.AuthenticatedAttributes.Add TempoSign
```

TIPI DI CERTIFICATI

Con Internet Explorer sono utilizzati due tipi di certificati: *personali* e di *sito Web*. Il primo tipo attesta l'identità di un utente ed è usato per inviare informazioni personali ad un sito, quando questo le richiede. Questi certificati per esempio possono essere richiesti dal sito di un'amministrazioni pubblica o di un ufficio postale. Il *"certificato di sito Web"*, invece, attesta l'autenticità e la protezione di un sito Web. Questi sono soprattutto utilizzati da siti che amministrano dati sensibili: banche, uffici postali, siti di commercio elettronica, ecc.

```
sig = Signedobj.sign(Signer, True)
Dim path As String
Dim pos As Integer
pos = InStr(CommonDialog1.filename,
                                    CommonDialog1.FileTitle)
path = Mid(CommonDialog1.filename, 1, pos - 1)
filename = path + "firmato" + CommonDialog1.FileTitle
Open filename For Output As #2
Write #2, sig
Close #2
MsgBox "Firmato il file: " + CommonDialog1.FileTitle
Set Signedobj = Nothing
Set Signer = Nothing
Set TempoSign = Nothing
Exit Sub
errore:
If Err.Number > 0 Then
   MsgBox "Errore VB " & Err.Description
Else
   MsgBox "Errore CAPICOM " & Err.Number
End If
End Sub
Function ScegliCertificato(obj As Signer) As Boolean
Dim objStore As New CAPICOM.Store
objStore.Open
 For i = 0 To List1.ListCount - 1
  If List1.Selected(i) Then
   obj.Certificate = objStore.Certificates.Item(i)
   ScegliCertificato = True
  End If
 Next i
 'bisogna inserire le istruzioni per verificare
 'la validità del certificato
End Function
```

Nella *Firma_Click*, dopo aver caricato in Signer il certificato selezionato sulla ListBox, si apre il file che si vuole firmare e il contenuto viene inserito nella stringa *cont*. Con l'istruzione successiva, utilizzando la proprietà *Content*, s'imposta l'oggetto *SignerData*. Utilizzando l'attributo, invece, si imposta l'ora e il giorno in cui viene firmato il documento, successivamente il valore dell'attributo viene impostato nella collezione di attributi del *Signer*. Infine viene firmato il documento attraverso il metodo sign dell'oggetto *SignedData*. Facciamo notare che, con la firma digitale, si esegue l'hash del dato da firmare, crittografandolo, usando la chiave privata contenuta nel certificato. Il secondo argomento del metodo *Sign* è *bDetached*, un valore che specifica (quando è *True*) che la firma e il dato da firmare non sono nello stesso documento; quindi, in questo caso, chi deve verificare la firma deve avere una copia del documento firmato. Invece, per verificare un documento firmato possiamo usare la seguente procedura.

```
Private Sub verificafirma_Click()
On Error GoTo error
Dim Cont As String
Dim ContFirmato As String
Dim Signdata As New SignedData
Dim path As String
Dim pos As Integer
pos = InStr(CommonDialog1.filename,
                                    CommonDialog1.FileTitle)
path = Mid(CommonDialog1.filename, 1, pos - 1)
filename = path + "firmato" + CommonDialog1.FileTitle
'apre file firmato
Open filename For Input As #1
Input #1, ContFirmato
Close #1
'apre file originale
Open txtfilename For Input As #2
Input #2, Cont
Close #2
Signdata.Content = Cont
On Error Resume Next
Signdata.Verify ContFirmato, True
If Err.Number <> 0 Then
 MsgBox "Errore nella verifica" & Err.Description
Else
 MsgBox "Verifica completata"
End If
Exit Sub
error:
If Err.Number > 0 Then
 MsgBox "Errore di VB: " & Err.Description
Else
 MsgBox "Errore di CAPICOM: " & Hex(Err.Number)
End If
End Sub
```

La procedura precedente verifica la correttezza della firma, nell'ipotesi che il contenuto del documento e la firma non siano nello stesso file (cioè il parametro *bDetached=True*). Infatti, vengono caricati due file, quello da firmare e la firma. Il primo è caricato in *Signdata.Content*, il secondo nella variabile *ContFirmato* che successivamente viene passata al metodo *Verify* dell'oggetto *SignedData*. Per eseguire queste operazioni correttamente il certificato deve essere attendibile (cioè installato nell'archivio delle autorità di certificazione).

CONCLUSIONI

In questo articolo abbiamo introdotto le tecnologie alla base dell'infrastruttura a chiave pubblica (PKI), fondamentale per le applicazioni che prevedono un sistema di protezione distribuito, in cui i partecipanti non fanno parte della stessa rete. Nel CD troverete degli esempi su come crittografare e decrittare un dato, e il codice degli esempi (algoritmo di hash) che abbiamo presentato nel precedente appuntamento.

Massimo Autiero

Per saperne di più sui certificati digitali potete collegarvi ai seguenti siti: http://thawte.ascia.net (trovate informazioni in italiano) e http://www.verisign.com. Quest'ultimo è il sito di una delle più importanti autorità di certificazione mondiali. Se invece cercate più informazioni su ASN.1 collegatevi a http://www.rsa.com.

Microsoft Solutions Framework 3.0: principi fondamentali

Otto regole per il successo dei progetti

Il Solution Framework proposto da Microsoft si basa su otto semplici principi, dettati dall'esperienza e dal buon senso, che ci guidano nello sviluppo di soluzioni anche molto complesse.

Abbiamo già parlato di MSF v3.0 nel numero di Gennaio 2004 di ioProgrammo. Abbiamo visto come MSF ci guida nella definizione del team di lavoro *(Team Model)*, nella gestione del processo *(Process Model)*, e che contiene delle discipline per la gestione del progetto *(Project Management)*, dei rischi *(Risk Management)* e delle competenze *(Readiness Management)*. In questo articolo vedremo invece gli otto principi fondamentali di MSF. Questi principi non sono specifici di MSF, ma possono essere applicati anche ad altre metodologie, ad esempio RUP o le metodologie agili.

SUDDIVIDETE LE RESPONSABILITÀ, DEFINITE I COMPITI

Ogni persona del team in ogni ruolo è responsabile della riuscita del progetto. Il progetto non può funzionare se i membri del team non collaborano nel gestire la responsabilità. Questo è contrario al credere comune che il project/product manager deve "comandare" per la riuscita del progetto. Se i membri del team non collaborano, il progetto è destinato al fallimento. Inoltre, se qualcuno si accorge che qualcosa non va, è suo preciso compito fare di tutto per cercare di risolvere il problema, in collaborazione con gli altri. Se tutti fossero responsabili di tutto, il progetto sarebbe nel caos. Per questo, assieme al principio di suddivisione delle responsabilità, c'è il principio della chiara definizione dei compiti. Questo principio stabilisce che ci devono essere delle chiare figure di riferimento per i vari ruoli (sviluppo, test, user interface, etc…), figure che devono diventare i punti di riferimento per capire come sta andando il progetto. Questo principio è alla base del *Team Model* di MSF, dove ogni ruolo ha pari responsabilità ma compiti distinti e ben precisi. Ogni gruppo non è un isola indipendente: ognuno ha una visione su tutto il progetto.

"POTENZIATE" I MEMBRI DEL TEAM

In un team di successo, tutti i membri hanno "il potere", temporaneamente e nelle aree in cui sono particolarmente portati. Se ognuno si impegna ad ottenere il massimo e a dare il massimo, gli altri faranno lo stesso. Certamente anche team gestiti in maniera autoritaria dai capi progetto possono riuscire a portare a termine i progetti, ma al termine del progetto i membri del team non saranno cresciuti, e soprattutto il vero potenziale della soluzione non sarà venuto alla luce, in quanto le persone non avranno avuto la possibilità di esprimere al massimo le loro potenzialità. L'attività di controllo è distribuita, e non è intesa come un'attività "poliziesca" da parte del capo progetto.

"CONCENTRATEVI" SUL BUSINESS VALUE

Questo principio è molto semplice. Bisogna concentrarsi su quello che serve per massimizzare il valore del progetto. Bisogna concentrarsi su quello che serve per massimizzare il valore della soluzione. Attività che non hanno uno scopo preciso di "business", che non forniscono valore aggiunto ai clienti, (come dover utilizzare a tutti i costi una tecnologia inutile ma "alla moda"), portano sempre problemi, da slittamenti delle date di consegna al mancato raggiungimento degli obiettivi minimi, fino alla cancellazione del progetto. I ruoli *User Experience* e *Product Manager* rappresentano gli interessi dei clienti e degli utenti all'interno dei progetti. Inoltre il *Process Model* di

• **MICROSOFT SOLUTIONS FRAMEWORK V3.0**
ioProgrammo
Gennaio 2004
http://www.ioprogrammo.net/?s=articoli&idc=17&idu=52

• **MSF WHITEPAPERS disponibili sul sito**
http://www.microsoft.com/msf

• **MSF: A POCKET GUIDE disponibile su**
http://en.itsmportal.net/goto/literatuur/boek/201.xml

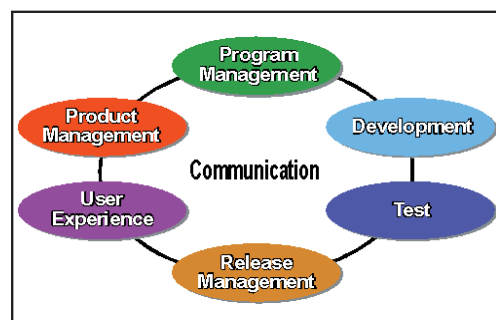



Fig. 1: Team Model di MSF – Un team di "pari".

MSF comprende anche la fase di deployment della soluzione perché una soluzione fornisce valore di business solo quando è completamente funzionante e utilizzabile dagli utenti.

LAVORATE PER UNA VISIONE COMUNE

I membri del team non possono lavorare bene se non condividono una visione comune. La visione è un documento molto breve, non più di qualche paragrafo, che contiene delle frasi chiare su quello che deve essere realizzato e sul Business Value della soluzione. Senza una visione, comune i membri del team, i clienti e gli utenti potrebbero trovarsi in disaccordo, durante lo svolgimento del progetto, sui punti cardine. Il documento di Vision aiuta a mantenere chiara la strada. La visione non scende dal cielo, ma i membri del team, devono lavorarci assieme a clienti ed utenti, per identificarla e condividerla. La visione comune è ciò che guida i membri di un team MSF, è quello verso cui tutti devono tendere, e che serve ad armonizzare gli sforzi, le competenze, i compiti e le responsabilità.

Fig. 2: ***Process Model di MSF – Fasi principali.***

RIMANETE "AGILI", ASPETTATEVI IL CAMBIAMENTO

Alla base di questo principio vi è il fatto che le cose cambiano. Il problema del modello di sviluppo a cascata è che si basa sul fatto che i presupposti e i requisiti del progetto non cambiano una volta stabilizzati. Questo non è vero neanche nei progetti più semplici. Per questo il *Process Model* di MSF prevede uno sviluppo basato su *Release* e *Milestone,* in cui il progetto può adattarsi all'evoluzione del contesto e dei requisiti. Il fatto di rilasciare spesso porta ad avere una conoscenza maggiore del sistema e degli impatti di possibili modifiche.

SUL WEB

Sul blog dell'autore è disponibile una lista di tutte le risorse disponibili su MSF, continuamente aggiornata. La lista (in inglese) è disponibile all'indirizzo: http://weblogs.asp.net/lbarbieri/articles/MSF_Resources.aspx

INCORAGGIATE UNA COMUNICAZIONE "APERTA"

La maggior parte dei fallimenti nei progetti è dovuta alla mancanza di comunicazione nel team e tra il team e i manager. Generalmente, se un progetto fallisce non è una sorpresa per i membri del Team, che sanno come va il progetto, ma è una sorpresa per quei manager che non sanno ascoltare. La comunicazione deve essere aperta anche agli utenti, ai clienti, e a tutte le persone interessate al progetto. Il *Team Model* di MSF è basato sulla comunicazione tra i membri del team, e i ruoli di *Program Manager, User Experience, Release Manager* e *Testing* hanno anche il compito di fungere da interfacce verso i clienti, gli utenti, gli amministratori di rete, l'help desk, etc.

L'AUTORE

Lorenzo Barbieri è laureato in Ingegneria Informatica e lavora come *Trainer* e *Consulente sulle tecnologie Microsoft.* È *Microsoft Certified Trainer* ed è certificato *MSF Practitioner.* Può essere contattato attraverso il suo blog: http://weblogs.asp.net/lbarbieri

IMPARATE DA "OGNI" ESPERIENZA

Bisogna imparare sia dai progetti e dalle attività andate a buon fine sia, e soprattutto, da quelle andate male. Non si può imparare dalle proprie esperienze se si è sempre di corsa, e se non si ha il tempo di analizzare l'andamento del progetto. In MSF dopo ogni *Milestone* ci sono sempre delle review pianificate di come è andata la *Milestone,* che culminano con il documentare cosa è andato bene e cosa è andato male, per utilizzarlo poi per la pianificazione delle *Milestone* o dei progetti futuri. L'imparare da "ogni" esperienza è quello che rende MSF un framework e non una metodologia. In MSF non esiste "la" strada da seguire, ma la strada il team la deve trovare applicando a MSF le proprie esperienze e il proprio modo di lavorare, cercando di ottenere il meglio.

INVESTITE NELLA QUALITÀ

La qualità deve essere un concetto sempre presente in tutte le fasi del progetto. Investire per avere dei Tester e delle persone addette alla User Experience durante l'analisi e il design del sistema, e non solo durante e dopo l'implementazione, porta a una migliore qualità dei requisiti, ad un design più adatto a essere testato, a un'interfaccia utente migliore. Tutti gli elaborati previsti dalle varie fasi vanno testati, non solo il programma. Questo comporta la revisione periodica dei documenti, per mantenerli aggiornati e per adattarli a cambiamenti improvvisi. La presenza poi nel Process Model della fase di *Stabilizzazione,* dove non si implementano nuove funzionalità ma si sistemano i problemi presenti, la misurazione del numero di bachi aperti e risolti, la gestione del deployment fin dalle prime release, porta ad un notevole incremento della qualità.

CONCLUSIONI

Se qualche principio non vi trova favorevoli, è meglio individuare subito quali parti di MSF applicano quel principio per capire se quelle parti di MSF fanno per voi oppure no. Tutti questi principi hanno provato sul campo la loro utilità, e molti di loro sono interdipendenti. Applicandoli ai vostri progetti otterrete sicuramente dei risultati che tenderanno a migliorare continuamente nel tempo.

Lorenzo Barbieri

I vantaggi dell'AOP nelle applicazioni Java

Turbo Java, grazie ad AspectJ

Uno degli utilizzi che rivela appieno le potenzialità di AOP nello sviluppo delle applicazioni è il caching dei dati, che consente di ottimizzare le prestazioni del software.

L'analisi delle performance delle applicazioni porta spesso ad evidenziare come determinate operazioni, particolarmente onerose in termini di tempo, siano ottimizzabili tramite l'uso di tecniche di caching. Si tratta solitamente di operazioni che richiedono accesso a database, a servizi remoti, oppure che consistono in elaborazioni particolarmente complesse, come avviene nel caso che è oggetto di studio in questo articolo: la trasformazione di documenti xml tramite xsl. L'approccio tradizionale a questo problema consiste nello sviluppare, o acquisire, un modulo di cache general-purpose e di gestire poi applicativamente le operazioni di caching dove necessario. Ciò si traduce in concreto nel dover inserire nel codice della propria applicazione, prima dell'invocazione delle funzioni, un controllo atto a verificare che non sia già stata eseguita in passato la medesima elaborazione, e la logica necessaria a memorizzare i risultati. Se questo tipo di soluzione è capace di indirizzare correttamente il problema del caching, un'analisi critica ci fa osservare che è necessario codificare esplicitamente nel codice applicativo tutta la logica necessaria per reperire e magazzinare valori nella cache. Questo fatto è di per sè negativo, perché dà origine a codice 'misto' (si parla di *tangled code*) in cui si mescola la logica di business con aspetti non funzionali; il risultato è una soluzione non flessibile, che non ci consente di applicare o rimuovere la funzionalità di caching dall'applicazione se non a prezzo di cambiamenti del codice.
La soluzione ideale, in linea di principio, dovrebbe consentire di codificare, in modo separato dalla logica di business, il meccanismo di caching e poi applicare in modo dichiarativo (cioè non all'interno del codice, ma in elementi separati) questo meccanismo nei punti dove vogliamo metterlo in atto. Per arrivare a questa soluzione ottimale, la programmazione ad oggetti non ci fornisce strumenti adeguati; è necessario un cambiamento radicale di paradigma, cambiamento reso possibile dalle *Aspect Oriented Programming*.

AOP, UN APPROCCIO ALTERNATIVO

L'Aspect Oriented Programming (AOP) è un paradigma di programmazione emergente che si contrappone e completa il tradizionale paradigma ad oggetti. Infatti, mentre quest'ultimo si propone di modellizzare le applicazioni attraverso l'uso di oggetti, entità dotate di dati e comportamenti, l'AOP si preoccupa di modellare i *crosscutting concern*, ovvero tutte quelle funzionalità che investono e sono utilizzate in modo trasversale in molti punti dell'applicazione. Esempi tipici di crosscutting concern sono il logging, la misurazione delle performance, i controlli di autorizzazione, la gestione delle transazioni e, naturalmente, il caching dei dati. AOP offre la possibilità di modellare separatamente dal codice applicativo i crosscutting concern, consentendo di aumentare la modularità del software e permettendo di aggiungere dichiarativamente (cioè all'esterno del codice applicativo) funzionalità ulteriori al software sviluppato. Essendo l'Aspect Oriented Programming un paradigma differente dalla programmazione ad oggetti, nel suo studio si incontra una terminolgia differente. Si parla quindi di

- **Join point**, cioè 'punti' all'interno del flusso di un programma in cui possono essere inseriti dei crosscutting concern. La granularità di tali punti è variabile: si parla di esecuzione di un metodo, di utilizzo di un oggetto, di gestione di una eccezione.
- **Advice**, ovvero il codice che definisce il crosscutting concern.

- **Aspect**, l'unione di un advice con uno o più join point. L'aspect definisce quindi interamente l'applicazione di un crosscutting concern all'interno di un programma (ovvero l'unione del codice da eseguire e i punti dove deve essere eseguito).

Per applicare l'Aspect Oriented Programming allo sviluppo del software è necessario dotarsi di strumenti appositi, in quanto i tradizionali compilatori non supportano questa modalità di lavoro; i tool che permettono di applicare AOP allo sviluppo software possono lavorare secondo due modalità:

- *post-processando* il codice compilato; dopo aver generato gli eseguibili, questi sono processati dal compilatore Aspect Oriented che aggiunge i crosscutting concern direttamente all'interno degli eseguibili
- eseguendo il *weaving* a runtime, ovvero eseguendo i crosscutting concern quando necessario senza la necessità di processare gli eseguibili

Al momento esistono implementazioni di compilatori Aspect Oriented per molti linguaggi; tra questi, Java offre una certa possibilità di scelta, con tre implementazioni particolarmente interessanti: AspectJ, JBossAOP e AspectWerkz. In questo articolo ci soffermeremo in particolare su AspectJ che, tra quelli citati, è sicuramente il più documentato e conta la comunità più vasta.

ASPECTJ: CONCETTI FONDAMENTALI

AspectJ è un progetto volto ad adottare i benefici dell'Aspect Oriented Programming nella programmazione in Java. Per la definizione di *aspetti, join point* e *advice*, AspectJ utilizza una sintassi propria, per certi versi simile a Java, ma che necessita di un certo periodo di apprendimento per essere padroneggiata; tuttavia già con poche istruzioni si possono costruire delle soluzioni efficienti. Pur esistendo dei buoni testi sull'argomento, sul sito è disponibile una *'The AspectJ Programming Guide'* che offre una spiegazione molto dettagliata della sintassi e di tutte le funzionalità utilizzabili, corredata da esempi di codice. In sintesi, per descrivere con AspectJ un crosscutting concern e la sua applicazione all'interno di un programma è necessario scrivere un *aspect;* l'aspect è l'entità principale (assimilabile alla classe della programmazione ad oggetti) entro la quale sono sempre presenti uno o più *poincut*, combinazioni di join point che definiscono punti nell'esecuzione dell'applicazione e uno o più *advice*, che definiscono i comportamenti da adottare in corrispondenza dei pointcut.

JOIN POINT E POINTCUT

AspectJ offre un set completo di join point predefiniti, che esprimono vari punti nell'esecuzione del codice in cui verranno poi inseriti i crosscutting concern; gli esempi a seguire faranno riferimento alla classe *Point* così definita:

```
public class Point
{ int x;
  int y;
  public void setX(int newX) { x = newX; }
  public int getX()  { return x;}
  public void setY(int newY) { y = newY; }
  public int getY()  { return y;} }
```

La sintassi per dichiarare i più comuni join point è la seguente

- l'oggetto corrente è di tipo *SomeType*, espresso con *this(someType)*
- esecuzione di un metodo, *execution(void Point-.setX(int))*
- chiamata di un metodo, *call(void Point.setX(int))*
- l'oggetto destinatario dell'elaborazione è di tipo *SomeType, target(someType)*
- è in esecuzione un exception handler, *handler(ArrayOutOfBoundsException)*
- gli argomenti di una chiamata a metodo sono *x1, x2* e *x3 args(x1,x2,x3)*

È sempre possibile, nella definizione di join point, utilizzare wildcard per poter dichiarare in modo conciso una serie di punti nel codice.
Volendo ad esempio indicare l'invocazione di qualsiasi metodo getter della classe *Point* scriveremo

Fig. 1: **Logica di caching dei risultati di una elaborazione.**

```
call(* Point.get*(..))
```

In questa espressione notiamo l'uso della wildcard '*' che indica genericamente tutti i possibili valori assumibili nel contesto (quindi tutti i tipi ritornabili e tutti i metodi che iniziano per *get*), e della wildcard '..' che esprime come siano da considerare i metodi con qualsiasi numero e tipo di argomento. I join point costituiscono i mattoni fondamentali con cui è possibile, con AspectJ, definire i *pointcut.* I pointcut sono combinazioni logiche dei join point che definiscono i punti di esecuzione del codice in cui il tool inserirà gli advice. Una dichiarazione di pointcut è la seguente:

```
pointcut setter(): target(Point) && (call(void
                          setX(int)) || call(void setY(int)));
```

dove *pointcut* è la parola chiave riservata per indicare la definizione di un pointcut, *setter()* è l'identificativo, e la parte che segue i due punti è la definizione espressa in termini dei joinpoint fondamentali. Nel caso sopra esposto, il pointcut si potrebbe esprimere come 'tutte le chiamate dei metodi *void setX(int)* e *void setY(int)* che operino su un oggetto di tipo *Target*'. Per garantire maggiore flessibilità ed ampliare gli scenari di utilizzo è possibile parametrizzare i pointcut; ciò, così come avviene per i metodi Java, ha lo scopo di rendere disponibile all'interno dell'advice alcuni valori che derivano dall'applicazione. Supponiamo, riprendendo l'esempio precedente, di voler rendere disponibile l'oggetto *Point*. Scriveremo quindi

```
pointcut setter(Point p): target(p) && (call(void
                          setX(int)) || call(void setY(int)));
```

È importante notare che solo tre tipi di joinpoint fondamentali possono essere utilizzati per esporre oggetti come parametri: *this, target* e *args*. Volendo quindi esporre come parametro anche l'*intero* passato nei setter scriveremo

```
pointcut setter(Point p, int i): target(p) && args(i) &&
                   (call(void setX(int)) || call(void setY(int)));
```

ASPECTJ
Inizialmente sviluppato nel 1997 dal team Xerox PARC guidato da Gregor Kiczales (ancora oggi figura di riferimento del settore), ha cominciato a godere di una fama notevole negli ultimi anni con l'ingresso del progetto nel consorzio open source Eclipse. È un tool liberamente scaricabile dal sito http://eclipse.org/aspectj/ ed è integrabile in molti ambienti di sviluppo, primo tra tutti Eclipse.

DEFINIAMO I COMPORTAMENTI: GLI ADVICE

Fig. 2: *Schermata di Ajdt, l'ambiente di sviluppo integrato Aspectj-Eclipse.*

Dato un *pointcut*, si possono scrivere diversi *advice* che definiscano i comportamenti che vogliamo inserire. Ancora una volta, per definire un advice, è necessaria una sintassi particolare che ci permetta di caratterizzare meglio il posizionamento degli advice rispetto al pointcut.

Ecco vari esempi di dichiarazioni di advice che utilizzano come riferimento l'ultimo pointcut esposto:

- *before(Point p, int i) : setter(p,i) {...}* specifica un advice da eseguire prima del pointcut *setter*
- *after(Point p, int i) : setter(p,i) {…}* specifica un advice da eseguire dopo il pointcut *setter*
- *around(Point p, int i) : setter(p,i) {…}* specifica un advice da eseguire 'attorno' il pointcut *setter*

In particolare, l'ultimo tipo di dichiarazione sarà utilizzata nell'esempio di caching che sarà esposto a breve. Il codice vero e proprio che descrive l'advice (abbreviato negli esempi precedenti in {...}) deve essere espresso in linguaggio Java, utilizzando eventualmente gli oggetti messi a disposizione come argomenti. Considerando il consueto esempio si potrebbe definire un advice così fatto:

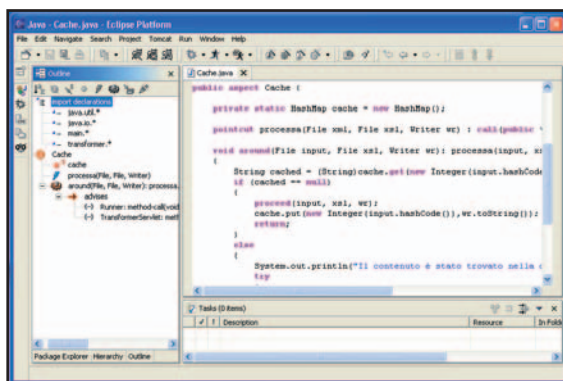
```
before(Point p, int i): setter(p, i)
{   System.out.println("Viene invocato un setter");
    System.out.println("Le coordinate prima
        dell'esecuzione sono: "+p.getX()+","+p.getY());
    System.out.println("Il valore inserito dall'utente è "+i);
}
```

ASPECT

L'aspect, come detto in precedenza, è l'entità fondamentale che racchiude *pointcut* e *advice*. Non essendoci ulteriori indicazioni da dare in merito, di seguito è riportato l'esempio che racchiude il pointcut *setter* e l'advice esposto in precedenza.

```
public aspect LoggingDeiSetter
{  pointcut setter(Point p, int i): target(p)&& args(i)
          && (call(void setX(int))||call(void setY(int)));
   before(Point p, int i): tre(p, i)
   {  System.out.println("Viene invocato un setter");
      System.out.println("Le coordinate prima
          dell'esecuzione sono: "+p.getX()+","+p.getY());
      System.out.println("Il valore inserito dall'utente è "+i); }}
```

UTILIZZO DEL COMPILATORE ASPECTJ

Prima di procedere all'analisi dell'esempio principale proposto in questo articolo, è necessario dare qualche indicazione in più su come utilizzare AspectJ. Come detto in precedenza, AspectJ offre un compilatore Aspect Oriented per Java: questo significa che il codice della applicazione deve essere prima compilato con *javac* come avviene di solito, e quindi processato dal tool *ajc* fornito da AspectJ. Si faccia riferimento al sito per tutti i dettagli relativi ad *ajc*. Per godere appieno delle potenzialità di AspectJ è però necessario disporre di un ambiente di sviluppo integrato. A ciò provvede il progetto *Ajdt* (sempre parte del consorzio Eclipse) che integra AspectJ con l'IDE Eclipse, rendendo lo sviluppo agevole ed immediato. Grazie ad *Ajdt* è possibile compilare e

preprocessare il codice in un'unica passata, nonchè disporre di varie viste grafiche che aiutano a capire dove gli aspetti sviluppati influiranno nel comportamento del codice.

UN SISTEMA DI CACHING CON ASPECTJ

Veniamo ora all'esempio principale proposto in questo articolo. Supponiamo di avere una semplice applicazione di web publishing che provveda a servire dei contenuti tramite il web; i contenuti sono memorizzati in file xml e, prima di essere spediti al client, vengono trasformati in html applicando un foglio di stile xsl. La struttura dell'applicazione prevede due semplici classi:

- una servlet *TransformerServlet*, che riceve la richiesta dell'utente e identifica il contenuto richiesto
- una classe *XslTransformer* che riceve dalla servlet il nome del file xml da processare ed applica la trasformazione specificata nel file xsl di stile.

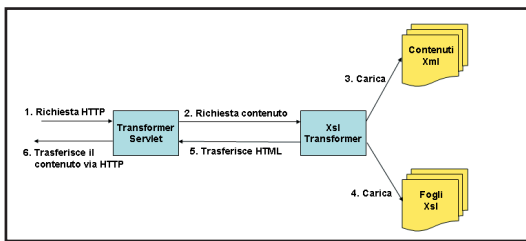



Fig. 3: ***Architettura della applicazione.***

Nella pratica si verifica che l'elaborazione xsl/xml è un processo poco performante, per cui è sensato pensare di dotare l'applicazione di un sistema di caching che, prima di processare un file xml, controlla se l'elaborazione è già stata eseguita in precedenza e, in caso affermativo, ripropone la pagina html già generata. Il vantaggio di implementare con AspectJ questo requisito, come si vedrà in seguito, sta nel fatto che il codice dell'applicazione pre-esistente non viene modificato; verrà semplicemente descritto un aspetto ed applicato tramite il compilatore Aspect Oriented al codice esistente. Vediamo innanzitutto le due classi che compongono l'applicazione; la classe *XslTransformer* utilizza le API *javax.xml* per effettuare l'elaborazione dei file xml:

```
public class XslTransformer
{  /**
    * Applica la trasformazione definita in fileXsl al
                          contenuto di fileXml e invia il
    * risultato della trasformazione a writer
    */
   public void process(File fileXml, File fileXsl, Writer
                     writer) throws TransformerException
```

```
{  Templates template = TransformerFactory.newInstance(
                ).newTemplates(new StreamSource(fileXsl));
   Transformer transformer = template.newTransformer();
   transformer.transform(new StreamSource(
                  fileXml), new StreamResult(writer)); }
}
```

La servlet principale che gestisce il dialogo con gli utenti non è riportata per intero nell'articolo, ma può comunque essere trovata nel CD allegato alla rivista. In sintesi, il suo compito è ricavare dalla request dell'utente il contenuto richiesto e fornirlo, dopo aver trasformato il file xml sorgente. Il metodo della servlet che esegue questa funzionalità è il seguente:

```
/**
 * Invia il contenuto richiesto dopo aver applicato
                   la trasformazione specificata dal
 * file XSL_FILE
 */
public void inviaContenuto(String fileXml,
     HttpServletResponse response) throws Exception
{  long time = System.currentTimeMillis();
   StringWriter sw = new StringWriter();
   XslTransformer t = new XslTransformer();
   t.process(getFile(fileXml), getFile(XSL_FILE), sw);
   response.getWriter().print(sw.toString());
   time = System.currentTimeMillis() - time;
   System.out.println("Contenuto " + fileXml + "
                   reso in " + time + " millisecondi");
}
```

Queste due classi, assieme al descrittore della web application creata (*web.xml*) in cui sia dichiarata e mappata correttamente la servlet, costituiscono un sistema di web publishing, minimale ma funzionante. Per testarlo è necessario effettuare il deploy della web application presente nel CD allegato in un servlet container (Jakarta Tomcat ad esempio) e quindi inviare una richiesta tramite un browser all'url *http://localhost:8080/publisher/provider?content=mypage*. Il risultato sarà simile a quello mostrato in Fig. 4. L'applicazione inoltre scrive nella console alcuni dati interessanti, fra cui spicca il tempo (in millisecondi) necessario ad eseguire la trasformazione xml/xsl. Applicando quanto illustra-

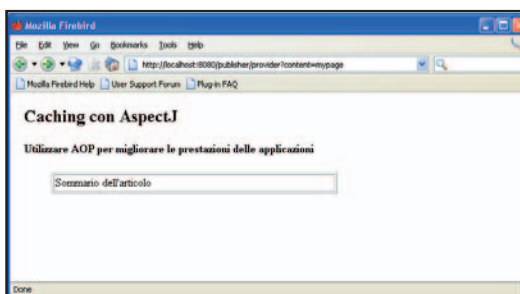



Fig. 4: ***Il sistema di web publishing in azione.***

TOOL ALTERNATIVI AD ASPECTJ

AspectWerkz
http://aspectwerkz.codehaus.org/ è un tool open source che fa parte della famiglia di progetti ospitati dal *Codehaus* http://www.codehaus.or. Il suo punto di forza è la capacità di effetturare il weaving degli aspetti a runtime, oltre che il post-processamento degli eseguibili.

JBossAOP
http://www.jboss.org/developers/projects/jboss/aop è invece un sottoprogetto del gruppo JBoss utilizzabile anche separatamente dall'omonimo application server.

to in precedenza vogliamo ora sviluppare un aspetto che permetta di effettuare il caching dei risultati delle operazioni di rendering al fine di ottimizzare i tempi di risposta a richieste ripetute di contenuti.
Per prima cosa si tratta di definire il pointcut, che in questo caso possiamo indentificare come 'la chiamata del metodo *process* della classe *XslTransformer'*; utilizzando la sintassi di AspectJ questo si esprime con

```
pointcut processa(File xml, File xsl, Writer wr) :
        call(public * XslTransformer.process(..)) &&
                                args(xml,xsl,wr);
```

Come si nota gli attributi *File xml, File xsl, Writer wr* sono stati esposti nel pointcut in modo da rendere disponibili i relativi valori nello sviluppo dell'advice. Quest'ultimo utilizza una dichiarazione di tipo *around*, il che sta a significare che intercetta l'esecuzione del programma prima e dopo il pointcut.
In un around advice il superamento del pointcut deve essere invocato esplicitamente con l'istruzione *proceed*.

```
void around(File input, File xsl, Writer wr):
                        processa(input, xsl, wr)
{  String cached = (String)cache.get(new
                    Integer(input.hashCode()));
   if (cached == null)
   {  proceed(input, xsl, wr);
      cache.put(new Integer(
            input.hashCode()),wr.toString());
      return;
   }
   else
   { System.out.println("Il contenuto è stato trovato
                                  nella cache");
      try
      { wr.write(cached); }
      catch (Throwable t)
      { t.printStackTrace(); }
      return;
   }
}
```

Questo advice utilizza una semplice logica di caching:

- dato un file, ne calcola l'*hashCode* e utilizza questo valore come chiave di ricerca in una *HashMap* che memorizzati tutti i file xml in precedenza trasformati
- se il valore dell'*hashCode* non è individuato tra i dati memorizzati invoca l'esecuzione del pointcut con l'istruzione *proceed*, e quindi memorizza nella cache la coppia (*hashCode*, file trasformato)
- se invece il valore dell'*hashCode* è individuato tra i dati memorizzati non viene eseguito il *pointcut* ma direttamente scritto il valore in memoria.

Dopo aver eseguito la compilazione con *ajc* dell'aspetto comprendente il poincut e l'advice definiti in precedenza, la riesecuzione dell'applicazione mostra un sensibile miglioramento delle prestazioni del sistema di web publishing quando sono presenti richieste ripetute di contenuti; dai log sulla console si può apprezzare come il tempo richiesto per per servire un contenuto presente in cache è pari a pochi millisecondi, mentre una trasformazione xml/xsl richiede un tempo nell'ordine dei decimi di secondo.
Se questo risultato non giunge inaspettato, è bene sottolineare ancora una volta il fatto che non è stato modificato il codice applicativo del sistema di web publishing, ma solo aggiunto un aspetto. Inoltre è possibile pensare a diverse migliorie al meccanismo di caching, in modo da rendere il sistema più flessibile e potente; in particolare, è facile generalizzare l'aspetto in modo tale da poter memorizzare qualsiasi tipo di dato, ottenendo così un meccanismo di caching applicabile in diversi punti del programma e completamente riusabile.

CONCLUSIONI

In questo articolo è stata esposta in estrema sintesi una introduzione all'Aspect Oriented Programming e ad AspectJ, un tool che permette di applicare l'AOP alle applicazioni Java.
Trattandosi di un paradigma relativamente giovane, le possibilità che esso può introdurre nello sviluppo del software non sono ancora del tutto esplorate. Tuttavia esistono già degli "aspetti standard" che vengono implementati e impiegati nelle applicazioni commerciali per risolvere in maniera efficiente ed elegante comuni crosscutting concern come logging, caching, gestione delle eccezioni e problematiche di autorizzazione.
Costruire un sistema di caching con AspectJ si è rivelato essere un lavoro abbastanza semplice, e ci ha consentito di aggiungere una nuova funzionalità ad una applicazione di web publishing senza modificare alcuna parte di codice scritta in precedenza.
Proprio la capacità di separare in moduli specifici funzionalità solitamente sparse in decine di punti nel codice applicativo si configura come la caratteristica "forte" dell'Aspect Oriented Programming che sta spingendo molti sviluppatori, ricercatori ed aziende a cercare di approfondire ulteriormente la materia e ad incorporare questo nuovo paradigma nei software dei prossimi anni.

Filippo Diotalevi

Salvare e leggere dati con Java 2

Un formato 'flessibile' per i file dati

Completiamo "Raffaello" con le funzioni *salva* e *apri* Implementeremo un pacchetto che gestisce con semplicità file in grado di creare automaticamente il proprio formato.

Questo mese, metteremo a punto gli ultimi dettagli di Raffaello, il programma di disegno che abbiamo creato nello scorso numero. Porremo la nostra attenzione sulle classi di gestione dei file disponibili con la JDK 1.4 e vedremo come usarle per creare un pacchetto che ci permetta di leggere e scrivere file in modo estremamente semplice e garantendo un comodo sistema di generazione automatica del formato dei dati, con particolare attenzione ai problemi legati alla compatibilità fra versioni diverse dello stesso programma.

Fig. 1: ***Il programma con le nuove funzioni, accessibili attraverso il menu Documenti.***

STRUMENTI NATIVI DI GESTIONE DEI FILE

Gli ideatori di Java hanno creato un sistema di gestione dei file basato su tre 'livelli', ai quali, nella JDK 1.4, ne è stato aggiunto un quarto.

Primo livello: gli oggetti *File*. Questi rappresentano un indirizzo nel file system. La classe *File* offre parecchi metodi che permettono di gestire molteplici file system con lo stesso codice.
Secondo livello: i *'flussi' ('stream')*, classi che gestiscono i comandi di sistema necessari per aprire e chiudere un file in lettura o scrittura. Noi useremo: *FileInputStream* e *FileOutputStream*. **Terzo livello:** i buffer, ingredienti necessari per 'sistemare' in memoria i dati, letti o da scrivere. Java ne mette a disposizione molti ma noi useremo solo il *ByteBuffer*. Infine abbiamo il nuovissimo **quarto livello:** il *'canale'* o *FileChannel*. Questo oggetto rappresenta la memoria locale del disco fisso, un buffer fisico dal quale passano tutti i dati in transito fra memoria centrale e file. L'introduzione di questo oggetto permette lo spostamento dei dati in piccoli gruppi lavorando fra buffer del disco e memoria centrale senza il costante coinvolgimento del supporto magnetico, riducendo il numero di interazioni con esso, che sono le operazioni più lente nella gestione dei file. Il sistema funziona in questo modo: si alloca un nuovo 'flusso' di dati con una *new FileInputStream (file);* e/o una *new FileOutputStream(file);* dove file è un *File*, l'indirizzo del file da utilizzare; se il flusso è stato allocato possiamo richiedergli il canale attraverso il metodo *getChannel*. Effettuate le operazioni di lettura e/o scrittura si dovranno chiudere canale e flusso usando i rispettivi metodi *close;* questo non solo libera le risorse ma effettua anche il flush del canale.

Fig. 2: ***Strumenti di gestione file ed "oggetti fisici" di riferimento.***

Nota bene: Lo stream viene allocato solo se il file esiste: per creare un file si usa il metodo *createNewFile* di *File*. Le operazioni di trasferimento dati si effettuano con la mediazione del buffer di memoria. Il *ByteBuffer* è un buffer di *byte* dedicato allo scopo. Il canale offre i metodi di lettura e scrittura *canale.-read(byteBuffer);* e *canale.write(byteBuffer);*. Esso funziona in questo modo: viene creato richiamando

il metodo statico *allocate(dimension)* dove *dimension* è la dimensione che dovrà avere. Ogni *ByteBuffer* ha due riferimenti interni: il primo *position* al momento della creazione punta all'inizio del buffer mentre il secondo *limit* si riferisce alla fine. Durante le operazioni di scrittura su buffer *position* avanza in modo da tenersi alla cella successiva all'ultima scritta. Conclusa l'operazione di scrittura bisogna utilizzare il metodo *flip* per riposizionare *position* all'inizio del buffer; contestualmente *limit* raggiunge la posizione successiva all'ultimo byte scritto. Ora è possibile leggere i dati contenuti fra *position* e *limit*. L'operazione descritta viene effettuata sempre: nelle operazioni di lettura, quando a riempire il buffer è il canale ed a leggerlo l'applicazione, e nelle operazioni di scrittura, allorché la situazione si capovolge.

Package it.gadget.fileFormat (classi pubbliche)

EFile
- canaleIngresso: FileChannel
- canaleUscita: FileChannel
- flussoUscita: FileOutputStream
- flussoIngresso: FileInputStream
- ingresso: File
- uscita: File
+ put(Bufferizzable): int
+ get(Bufferizzable): void
+ closeInput():boolean
+ closeOutput(): boolean
+ openInput(File): boolean
+ openOutput(File): boolean

interface
Bufferizzable
+ getFileFormatData:FileFormatData
+ beforeFileWriting: void
+ afterFileReading: void

FileFormatData
...
+ FileFormatData(String[], byte)
+ FileFormatData(String[], int, FileFormatData)

Fig. 3: **Oggetti ed interfacce pubbliche del pakage it.gatget.fileFormat.**

NOTA

POINT2D

Java definisce un tipo astratto *Point2D* estendendo il quale sono stati creati *Point2D.Double*, *Point2D.Float*. L'estensione della classe prevede l'implementazione del metodo *setLocation(double, double)*; e la definizione delle coordinate *x* ed *y*.

IL NOSTRO PACKAGE

Il nostro obiettivo principale sarà semplificare e la cosa più semplice che posso immaginare è una classe, che possiamo chiamare *EFile*, che mi fornisca i metodi *openInput(File)*, *closeInput()*, *openOutput(File)*, *closeOutput()*, *put(Object)* e *get(Object)*. I primi quattro si scrivono senza troppe difficoltà, gli ultimi due, semplicemente non si scrivono! È vero che Java garantisce un metodo nativo di scrittura di un oggetto su file, la cosiddetta "serializzazione", ma questo scrive su file tutti gli attributi dell'oggetto mentre noi vogliamo scegliere quali scrivere. Sarà necessaria un'interfaccia che ci fornisca dei metodi implementando i quali ogni oggetto possa 'prepararsi' per la scrittura e riorganizzare i dati letti oltre che scegliere cosa leggere e scrivere dal e sul file; siccome letture e scritture utilizzeranno un buffer, chiameremo l'interfaccia *Bufferizzable* ed i metodi di *EFile* saranno *put(Bufferizzable)* e *get(Bufferizzable)*. L'interfaccia avrà questa forma:

Fig. 4: **Formato del file proposto.**

```
public interface Bufferizable
{ public FileFormatData getFileFormatData();
  public void beforeFileWriting();
  public void afterFileReading(); }
```

FORMATO DI UN FILE

Il problema più grosso nella gestione delle informazioni contenute nei file è che, una volta trasferite sul supporto magnetico, queste perdono la loro 'identità': in altre parole non è più possibile distinguere il 'tipo' originale dei dati. Questo significa che: o si segue una regola di preventiva 'dichiarazione' del tipo di dato memorizzato, la serializzazione utilizza questa strategia ed anche XML può considerarsene un esempio evoluto di impiego, oppure si stabilisce in quale ordine i dati contenuti nei nostri oggetti-informazione debbano essere trasferiti sul disco dichiarando, in questo caso, quanti byte vengano utilizzati per ciascun blocco di dati. Il secondo sistema è il meno 'amichevole' ma il più 'rispettoso' del sempre esiguo spazio libero dei dischi fissi e quindi sarà il sistema proposto per Raffaello.

L'IDEA DELLA COMPATIBILITÁ

Il secondo problema che ci siamo posti è garantire la massima compatibilità fra file scritti da diverse versioni dello stesso programma. Per cominciare vediamo il meccanismo base di creazione del formato: nel nostro esempio dobbiamo salvare i dati relativi al nostro foglio, quindi quelli del disegno, ovvero quelli dei percorsi, cioè quelli dei punti.
Salvare i dati relativi al punto è piuttosto facile: si tratta delle due coordinate, due *double* di 8 byte ciascuno, e di un carattere, 2 byte, che distingua i punti normali da quelli di controllo. Il formato di memorizzazione del punto sarà dunque 18, (valore di *x*), (valore di *y*), (carattere), dove 18 è la dimensione del blocco dei dati da memorizzare. Ma se esistesse una versione zero di Raffaello capace di gestire solo spezzate? Allora un file scritto con tale versione conterrebbe punti di questo formato: 16, (valore di *x*), (valore di *y*). Non avrebbe avuto senso allora, l'aggiunta di un carattere al dato. Ciò nondimeno la nuova versione dovrebbe essere in grado di gestire l'informazione presente nei vecchi file. Le spezzate siamo in grado di gestirle! Per capire meglio dichiariamo un oggetto punto così come avremmo dovuto dichiararlo nella precedente e mai esistita versione del programma. Già che ci siamo: siamo sicuri che siano necessari dei *double* per le coordinate? I monitor ed i mouse non restituiscono coordinate con virgola ed un *double* è estremamente oneroso anche in termini di memoria utilizzata rispetto, ad esempio, ad un piccolo *short*. Chiamiamo la nuova (o vecchia?) classe *Point2dShort*.

```
package it.graphic;
import java.awt.Point;
import java.awt.geom.Point2D;
```

```
public class Point2dShort extends Point2D
{ short x=0, y=0;
  public Point2dShort(){}
  public Point2dShort(Point point)
  { setLocation(point.x, point.y); }
  public Point2dShort(Point2d point)
  { setLocation(point.x, point.y); }
  public void setLocation(double x, double y)
  { this.x=(short)x; this.y=(short)y;}
  public double getX() { return x; }
  public double getY() { return y; } }
```

Il formato di scrittura di questa classe sarà: 4, (valore di *x*), (valore di *y*) dato che la short occupa due byte. Ora modifichiamo *Point2d* facendogli estendere *Point2dShort* anziché *Point2D.Double*. Tutto funziona comunque, le coordinate *short* non creano problemi, ma come sarà ora il formato di *Point2d*? Per compatibilità dovrebbe essere: 4, (valore di *x*), (valore di *y*) ed anche a logica sembra corretto che lo sia, quando non è punto di controllo. Ma se è punto di controllo diventerà necessariamente 6, (valore di *x*), (valore di *y*), (carattere)! Schematizziamo la situazione in questo modo: la dimensione sarà sempre un numero e sempre dello stesso tipo, cambierà solo il valore, le coordinate sono sempre *short* e sono sempre due, il carattere può esserci o non esserci.
L'idea è quindi questa: creiamo un sistema che si preoccupi di recuperare il blocco di dati necessario nascondendo all'oggetto la dimensione dello stesso, quindi introduciamo sistemi di gestione diversi per i dati essenziali, le coordinate, ed i dati opzionali, il carattere. Allo scopo realizzeremo la classe *FileFormatData* che ci permetterà di dichiarare l'elenco degli attributi da leggere/scrivere ed un oggetto *FormatHandler* che gestirà le operazioni di lettura e scrittura sfruttando un riferimento all'oggetto; quest'ultima classe sarà estesa da *FileFormat* e *OptionalData* allo scopo di garantire la capacità di distinguere fra dati necessari e dati opzionali.

IL GESTORE DI FILE

Ora abbiamo gli ingredienti necessari per definire i metodi *get* e *put* di *EFile*.

```
public int put(Bufferizable object) throws Exception
{ object.beforeFileWriting();
   FileFormat ff=new FileFormat(
                        object.getFileFormatData(), true);
  ff.setReference(object);
  int dimBuffer=ff.onBufferSize(), dimDato=0;
  ByteBuffer b=ByteBuffer.allocate(dimBuffer);
  try
  { ff.toBuffer(b);
     b.flip();
     dimDato=canaleUscita.write(b); }
  catch (Exception e) { throw e; }
  return dimDato; };
```

Il metodo put richiede all'oggetto di prepararsi alla scrittura quindi crea *ff*, un oggetto *FileFormat*, dandogli come parametro il *FileFormatData* restituito dall'oggetto ed una variabile booleana indicante la necessità di un formato di scrittura. Dopo aver ottenuto il riferimento all'oggetto, *ff* diviene il solo interlocutore: esso restituisce la dimensione del buffer necessaria e copia i dati sul buffer; a questo punto, effettuata la flip possiamo scrivere sul file.

```
public void get(Bufferizable object) throws Exception
{ FileFormat ff=new FileFormat(
                        object.getFileFormatData(), false);
  ff.setReference(object);
  int dimDato=0, dimFormato=ff.getHeaderType();
  ByteBuffer b=null;
  try
  {  if (canaleIngresso.size()<dimFormato)
    { //lancia l'eccezione// }
    ByteBuffer bb=ByteBuffer.allocate(dimFormato);
    canaleIngresso.read(bb);
    bb.flip();
    switch (dimFormato)
    {  case FileFormat.BYTE: dimDato=bb.get(); break;
       case FileFormat.SHORT: dimDato=bb.getShort();
                                                 break;
       case FileFormat.INT: dimDato=bb.getInt(); break;}
    if (canaleIngresso.size()<dimDato)
    { //lancia l'eccezione// }
    b=ByteBuffer.allocate(dimDato);
    canaleIngresso.read(b);
    b.flip();
    ff.byBuffer(b); }
  catch (Exception e) { throw e; }
  object.afterFileReading(); }
```

Il metodo *get* è un po' più complesso: per cominciare crea *ff* indicando la necessità che sia un formato di lettura, quindi, dopo averlo dotato del riferimento all'oggetto si fa restituire la dimensione dell'intestazione, ovvero i byte spesi per memorizzare la dimensione del blocco dati. Con i controlli del caso, alloca il buffer necessario e legge l'intestazione dal file, quindi alloca il buffer per i dati e li legge dal file e, dopo la *flip*, chiede ad *ff* di estrarre la sua parte di informazione. Conclusa l'operazione ordina all'oggetto di riorganizzare i dati letti. I metodi *save* e *open* di *ActionHandler* saranno dunque di questo tipo:

Fig. 5: **Struttura del package fileFormat e relazioni coi package dell'SDK 1.4.**

FILEFILTER

FileFilter è definita in due pacchetti: *javax.swing.filechooser* e *java.io*. La prima è discussa nell'articolo ed ad uso esclusivo delle *JFileChooser*, la seconda è un'interfaccia che richiede l'implementazione del metodo accept (*File pathname*); metodo che restituisce un booleano. Per usarlo si pone l'indirizzo di una cartella in un oggetto *File* a cui si applica il metodo *listFiles* che prende il filtro come parametro. Il metodo restituisce l'array *File[]* dei file che si trovano nella cartella e verificano le condizioni imposte.

```
private void save(File output)
{  try
  {  EFile nuovo=new EFile();
     if (!nuovo.openOutput(output)) { // messaggio
                                          d'errore // }
    nuovo.put(gestoreFormato);
    nuovo.put(finestra.foglio);
    if (!nuovo.closeOutput()) // messaggio d'errore // }
  catch (FileExceptions fe) { // messaggio d'errore // }
  catch (Exception e) { // messaggio d'errore // } }
private boolean open(File input)
{  try
  { EFile nuovo=new EFile();
     ...
  return true;
}
```

L'oggetto *gestoreFormato* è di tipo *ProgramFormatHandler,* una classe bufferizzabile definita nel nostro pacchetto file per gestire i dati sulle versioni dei programmi scriventi e le compatibilità con le versioni precedenti. Il costruttore accetta tre parametri: versione e revisione dello scrivente e versione compatibile (vedi box). Nel nostro esempio abbiamo due possibilità:

1) autorizziamo la lettura dei file 'nuovi' anche con la vecchia versione: la vecchia versione, quella che non esiste, interpreterebbe i punti di controllo come punti normali;
2) vietiamo la lettura alla vecchia versione: l'istruzione *gestoreFormato.isCompatible()*; restituireb be false per indicare l'incompatibilità di versione.

RIFLESSIONE

I metodi di riflessione permettono di accedere a campi e metodi di una classe, pubblici, privati o protetti; la chiave del meccanismo risiede nella classe *Class* e nel fatto che, per assicurare il polimorfismo, il linguaggio tiene traccia dei metodi e degli attributi di ciascuna classe utilizzata. A questo punto è sufficiente un riferimento alla particolare istanza dell'oggetto per violare le regole di incapsulamento ed ottenere risultati quali il meccanismo di serializzazione o il nostro package di gestione file.

OGGETTI "BUFFERIZZABILI"

Vediamo qualche implementazione dei metodi previsti dall'interfaccia *Bufferizzable*. Allo scopo distinguiamo due casi:

1) l'oggetto è una classe bufferizzabile di base: un caso significativo è quello di *Path*.

```
private int coloreIntero, bordoIntero, sfondoIntero;
private char caratteristiche;
private Point2d[] vettorePunti;
private static String[] attributiDaScrivere={
        "coloreIntero", "bordoIntero", "sfondoIntero",
                 "caratteristiche", "vettorePunti"};
private static FileFormatData datiFormato=new
FileFormatData(attributiDaScrivere, FileFormatData.INT);
public FileFormatData getFileFormatData()
{ return datiFormato; }
public void afterFileReading()
{ colore=new Color(coloreIntero);
...
  modificato=true; }
public void beforeFileWriting()
{ coloreIntero=colore.getRGB();
  bordoIntero=bordo.getRGB();
  sfondoIntero=sfondo.getRGB();
  caratteristiche=(chiuso)?'c':'a';
  vettorePunti=punti.toArray();  }
```

Notiamo che fra gli attributi dichiarati abbiamo *attributiDaScrivere* che è un vettore di stringhe statico contenente i nomi degli attributi da leggere/scrivere ed un *FileFormatData,* anch'esso statico poiché i dati di formato saranno sempre quelli ed anche il tipo di variabile numerica usata come intestazione. I metodi più significativi sono *afterFileReading* e *beforeFileWriting* in cui si può vedere come i dati debbano essere riorganizzati. La ragione è semplice: né gli oggetti *Color* né gli oggetti *Vector* possono essere letti/scritti 'direttamente' su file, non essendo bufferizzabili. Inoltre ed il nostro *FormatHandler* è in grado di gestire solo oggetti bufferizabili, tipi fondamentali non booleani ed array, anche multidimensionali, di oggetti bufferizzabili o di tipi fondamentali non booleani.

2) l'oggetto estende una classe bufferizzabile, è il caso della nostra *Point2d:*

```
protected char caratteristiche='\0';
private static String[] attributiDaAggiungere={
                                   "caratteristiche"};
public FileFormatData getFileFormatData()
{  int i=(caratteristiche=='\0')?0:1;
   return new FileFormatData(attributiDaAggiungere,
                     i, super.getFileFormatData()); }
public void afterFileReading()
{ controllo=(caratteristiche=='c'); }
public void beforeFileWriting(){}
```

Qui i dati sono opzionali, quindi devono essere letti/scritti col metodo e con i controlli del caso, ed essendo da aggiungere ai dati della superclasse nella *getFileFormatData* costruiremo 'al volo' un *FileFormatData* che fornisca l'elenco dei dati opzionali, il numero di questi che dovranno essere scritti nel caso di una scrittura su file ed un riferimento al *FileFormatData* fornito dalla superclasse.

IL GESTORE DI FORMATO

Arriviamo al cuore del package: iniziamo con uno sguardo a *FileFormat*. Subito dopo spenderemo qualche parola attorno all'implementazione di *FormatHandler*. Non ci occuperemo di *OptionalData* data la sua semplicità. Per cominciare, *FileFormat* dichiara un *Vector* detto estensioni che raccoglierà

gli eventuali *OptionalData*.

```
private Vector estensioni=null;
```

Il costruttore scandisce *fileFormatData* creando tanti oggetti *OptionalData* quante sono le estensioni opzionali del formato, la prima estensione che dichiara almeno un attributo in scrittura forza le successive alla scrittura di tutti i loro attributi. La scansione finisce con l'individuazione del *FileFormatData* restituito dall'oggetto *Bufferizzable base*. I dati di questo definiscono il *FileFormat*. Il metodo *setReference* scandisce estensioni risalendo la catena di derivazione delle classi. A ciascuna estensione viene associato l'oggetto e la particolare classe nella catena di derivazione.

```
public void setReference(Object object) throws
                                FileFormatException
{ reference(object);
  Class c=object.getClass();
  int tot=(estensioni==null)?0:estensioni.size();
  for (int i=tot; —i>=0;)
  { FormatHandler fh=(FormatHandler)
                          estensioni.elementAt(i);
    fh.reference(object);
    fh.classeOggetto=c;
    c=c.getSuperclass(); }
  if (c==null) // gestione errore //
  classeOggetto=c; }
```

Il metodo *onBufferSize* restituisce la dimensione del blocco dati e dell'intestazione relativa all'oggetto a cui si riferisce il *FileFormat* in questione. Allo scopo si scandisce *estensioni*, sommando la dimensione dei dati di ciascuna a quella dei dati dell'oggetto *Bufferizzable* fondamentale. Il metodo *toBuffer* esegue il trasferimento dei dati sul buffer incominciando con l'intestazione, quindi con i dati propri, quelli essenziali, ed infine con i dati opzionali.

```
public boolean toBuffer(ByteBuffer b) throws FileExceptions
{  if (b.remaining()<dimBlocco) return false;
   switch (intestazione)
  {  case BYTE: b.put((byte)(dimBlocco-intestazione));
                                              break;
     case SHORT: b.putShort((short)(
                   dimBlocco-intestazione)); break;
     case INT: b.putInt(dimBlocco-intestazione); break;
     default: // gestione errore // }
  dataToBuffer(b);
  if (estensioni==null) return true;
  Iterator i=estensioni.iterator();
  while (i.hasNext())
  { OptionalData o=(OptionalData)i.next();
     o.dataToBuffer(b); }
  return true;
}
```

Infine, il metodo *byBuffer*, esegue la lettura dei dati dal buffer incominciando con i dati propri, quelli essenziali, per finire con i dati opzionali.

```
public boolean byBuffer(ByteBuffer b) throws Exception
{ dataByBuffer(b);
  if (estensioni==null) return true;
  Iterator i=estensioni.iterator();
  while (i.hasNext() && b.hasRemaining())
 { OptionalData o=(OptionalData)i.next();
    o.setDataAmount(OptionalData.REQUIRED);
    o.dataByBuffer(b); }
 return true; }
```

Riguardo *FormatHandler* diciamo subito che realizza i suoi metodi ricorrendo a metodi di riflessione. Tali meccanismi sono estremamente delicati poiché permettono di violare l'incapsulamento e, dato che questa strategia contravviene alla logica di Java, non approfondiremo l'argomento. La logica dei metodi fondamentali, *dataToBuffer, dataByBuffer* e *dataSize*, è quella di recuperare l'accesso agli attributi dell'oggetto utilizzando il loro nome. Recuperato l'accesso è possibile sapere se l'attributo sia un tipo fondamentale, un array o un *Bufferizzable* ed agire di conseguenza. Ad esempio, se l'attributo è un tipo fondamentale, si usa il metodo *primitiveToBuffer* per la scrittura:

Fig. 6: ***Se il nome di un campo è errato si ottiene un'eccezione al momento dell'accesso. Il package exceptions non sarà discusso.***

```
private boolean primitiveToBuffer(Object field, Class fieldClass)
{ ...}
```

in cui utilizziamo i metodi *put* del *ByteBuffer*, ed il metodo *primitiveByBuffer* per la lettura:

```
private boolean primitiveByBuffer(Field field, Object
                             object, Class fieldClass)
{...}
```

nel quale utilizziamo i metodi *get* del *ByteBuffer*.

CONCLUSIONI

Con questo è tutto. Ulteriori sviluppi sarebbero possibili: ad esempio aggiungere il supporto per i formati immagine standard (a riguardo rimando al tip apparso sul numero 80 della rivista) ma lo spazio è tiranno. Quindi vi lascio con la speranza che Raffaello possa divertirvi ed essere spunto per nuove idee.

Stefano Russo

SERIALIZZAZIONE

La serializzazione è un meccanismo, previsto dal linguaggio, di trasformazione di un oggetto e dei suoi attributi in una serie di byte e viceversa. Lo strumento è potente ma non risponde alle nostre necessità: i dati di formato, ad esempio, contemplano il nome della classe ed altre informazioni specifiche relative all'oggetto serializzato allo scopo di ricostruirlo tale e quale in lettura. Questo crea problemi di compatibilità qualora successive versioni del programma utilizzassero opportune estensioni degli oggetti già esistenti.

Applicazioni fisico-matematiche e di grafica 3D

Un motore di simulazioni fisiche

parte seconda

Consta di diverse componenti relazionate secondo una derivazione gerarchica. Le basi, i sistemi di coordinate e i corpi rigidi sono le classi per raggiungere l'obiettivo.

Dopo aver percorso i primi passi verso la realizzazione di un motore di simulazioni fisiche, attraverso l'implementazione di opportune classi di base per la manipolazione dei dati tipici dei fenomeni fisici, è arrivato il momento di fare un ulteriore cammino che porti alla definizione di una classe in grado di esprimere, da un punto di vista matematico, in modo completo, un corpo rigido. Ricordo, che la volta scorsa abbiamo già sviluppato due importanti oggetti *VECTOR* e *MATRIX* (questo ultimo da non confondere con il soggetto dei fratelli Wathoski!) con i quali abbiamo assolto il compito di base riferito all'analisi vettoriale, passaggio fondamentale e punto di riferimento per qualsiasi fenomeno si voglia descrivere in uno spazio 3D. Si tratta adesso sulla base dei metodi sviluppati di costruire nuove classi che consentano la produzione di un oggetto finale corpo rigido. Come vedremo, il passaggio non è immediato, anche se il percorso è rettilineo e non prevede particolari difficoltà se non qualcuna di ordine teorico che tenterò di non far apparire con opportuni chiarimenti, ma che comunque è facilmente colmabile dalla consultazione di un qualsiasi testo sulle nozioni di base della geometria lineare. Non lasciamoci ingannare dal titolo, i nuovi oggetti saranno anche gli ultimi (certo chi avesse voglia e tempo per farlo potrebbe, a partire dal materiale prodotto e che produrremo, sviluppare altri interessanti componenti). La piattaforma che si sta creando essendo ben strutturata e ricca di contenuti è un ideale punto di partenza per ulteriori sviluppi. Per concludere questa premessa vorrei ribadire come il programmatore ad oggetti non troverà alcuna difficoltà nella lettura di questo articolo proprio per la natura del progetto. Non sarà necessario ai fini della comprensione della presente trattazione conoscere i metodi matematici per la manipolazione di vettori e matrici, basterà sapere che essi esistono e che si possono modificare mediante operazioni come il prodotto vettoriale, la normalizzazione e quanto altro, senza peraltro avere il cruccio di conoscere il metodo usato per realizzare tali operazioni. Certo, per una completa e profonda conoscenza dell'argomento si auspica la presa visione di tali fasi dello sviluppo del progetto, ma come ripeto, ciò non è richiesto per la piena comprensione della seconda parte che ci apprestiamo a sviluppare.

UNA BREVE INTRODUZIONE

Le cosa importante da ricordare è che ci muoviamo in uno spazio a tre dimensioni tipico dei giochi 3D. È in questo ambito che intendiamo costruire dei modelli matematici successivamente implementabili come classi per lo sviluppo di un motore di simulazioni fisiche. Il linguaggio di programmazione usato è il C++ per la sua naturale propensione alla OOP, in particolare di tale linguaggio si apprezzano le caratteristiche proprie dell'ereditarietà e soprattutto il potente overloading che è possibile attuare. Mediante quest'ultimo, dopo aver superato il non banale ostacolo iniziale di progettazione e sviluppo di funzioni che prevedano la sovrapposizione di operatori come ad esempio la somma tra matrici, l'implementazione diventa del tutto naturale e può essere facilmente realizzata, come se scrivessimo tipiche espressioni di algebra lineare per la manipolazione di vettori e matrici, anche con le stesse notazioni. Ma cosa dobbiamo fare adesso? Sviluppare innanzitutto una classe di base (*BASIS*) in grado di descrivere un sistema orientato, e lo faremo definendo un sistema ortonormale. A tale classe assoceremo importanti metodi per la rotazione e la traslazione. Da essa potremo derivare un altro importante og-

CODICE SUL CD

Le classi sviluppate in ordine di apparizione sono contenute all'interno del CD nei file: *basis.cpp*, *coord.cpp* e *bodies.cpp*

getto che definisce uno o più sistemi di assi cartesiani *(COORD_FRAME)*, il che consente la descrizione di fenomeni su corpi in movimento. Si pensi alle dinamiche che possono avvenire all'interno di un treno in viaggio, è chiaro che prenderemo come riferimento lo scomparto dove ci troviamo, se si vuole conoscere il movimento di un oggetto è necessario tenere conto del fatto che il treno non è fermo. Infine, produrremo il corpo rigido *(RIGID_BODY)*. Le varie classi sono relazionate da una catena di derivazione gerarchica riportata in Fig. 1.

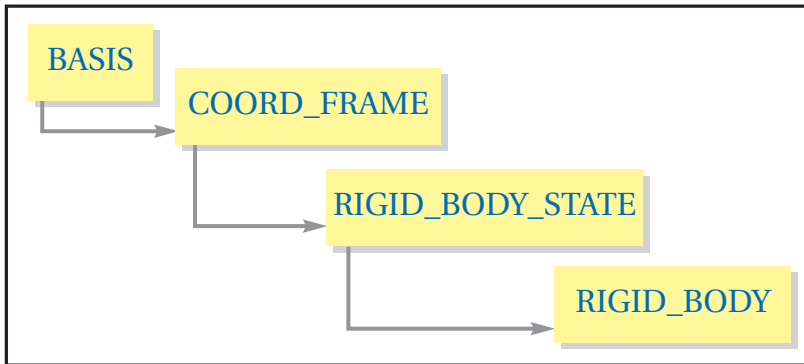



Fig. 1: ***Struttura ad oggetti del progetto, secondo le regole di derivazione.***

TIPI BASE

Un vettore orientato può essere memorizzato in più modi. Una prima soluzione prevede l'individuazione di una quaterna di numeri. Così facendo è sufficiente memorizzare quattro scalari, le operazioni (da quelle semplici come la somma e la differenza, alla rappresentazione grafica) in questo contesto richiederanno un'efficiente interpolazione. Una seconda soluzione fa uso di una base ortonormale, ovvero, tre vettori unità mutuamente ortogonali, descritti da una matrice di dimensione 3 x 3. In tal caso si usano ben nove scalari e l'interpolazione risulta meno efficiente, ciononostante è una soluzione appetibile; infatti, sarà la strada che intraprenderemo, poiché è facile la trasformazione da e per un altro sistema di assi cartesiani (manovra di uso comune sia nelle simulazioni fisiche che nei giochi 3D). Ci si potrebbe chiedere come mai vi è la necessità di definire una classe come base di rappresentazione di un vettore quando, come descritto, è sufficiente una matrice di dimensione 3 x 3. La risposta è indirettamente data dal rispetto delle norme della OOP che impongono una chiarezza semantica tra vari oggetti che hanno scopi diversi. Del resto, mentre per un vettore orientato sono richieste operazioni come la rotazione e la traslazione, per una matrice si hanno calcoli matematici di base come la somma e il prodotto.

```
// Base ortonormale
class BASIS
{ public:
  MATRIX R;
  public:
  BASIS()
  {}
  // Costruttore come tre vettori giustapposti
  BASIS(const VECTOR& v0,
                    const VECTOR& v1,const VECTOR& v2)
  // Implementato come matrice R (elemento MATRIX)
  : R( v0, v1, v2 )
  {}
  // Costruttore direttamente come matrice
  BASIS( const MATRIX& m )
  : R( m )
  {}
  // Operatori di overloading
  const VECTOR& operator [] ( long i ) const {
                                    return R.C[i]; }
  const VECTOR& x() const { return R.C[0]; }
  const VECTOR& y() const { return R.C[1]; }
  const VECTOR& z() const { return R.C[2]; }
  const MATRIX& basis() const { return R; }
  void basis( const VECTOR& v0, const VECTOR& v1,
                                 const VECTOR& v2 )
{this->R[0] = v0;
this->R[1] = v1;
this->R[2] = v2; }
// Rotazioni con la regola della mano destra
void rotateAboutX( const SCALAR& a )
{
  if( 0 != a ) // Non ruota se zero
  {
    VECTOR b1 = this->Y()*cos(a) + this->Z()*sin(a);
    VECTOR b2 = -this->Y()*sin(a) + this->Z()*cos(a);
    //cambio di base
    this->M[1] = b1;
    this->M[2] = b2;
    //x non cambia }
}
void rotateAboutY( const SCALAR& a )
{
  if( 0 != a ) // Non ruota se zero
 {
    VECTOR b2 = this->Z()*cos(a) + this->X()*sin(a);
                                          //ruota z
    VECTOR b0 = -this->Z()*sin(a) + this->X()*cos(a);
                                          //ruota x
    //cambio di base
    this->M[2] = b2;
    this->M[0] = b0;
    //y non cambia }
}
void rotateAboutZ( const SCALAR& a )
{ if( 0 != a ) // Non ruota se zero
  {  VECTOR b0 = this->X()*cos(a) + this->Y()*sin(a);
                                          //ruota x
     VECTOR b1 = -this->X()*sin(a) + this->Y()*cos(a);
                                          //ruota y
```

```
    //cambio di base
    this->M[0] = b0;
    this->M[1] = b1;
    //z non cambia }
}
//rotazione delle basi intorno l'unita di componente u
                                    dell'angolo theta (radianti)
void rotate( const SCALAR& theta, const VECTOR& u );
//rotazione di lunghezza da e angolo theta,
                                    nella direzione di u
void rotate( const VECTOR& da );
// Trasformazioni
const VECTOR transformVectorToLocal( const
                                    VECTOR& v ) const
{ return VECTOR( R.C[0].dot(v), R.C[1].dot(v),
                                    R.C[2].dot(v) ); }
const POINT transformVectorToParent( const
                                    VECTOR& v ) const
{ return R.C[0] * v.x + R.C[1] * v.y + R.C[2] * v.z; }
};
```

Come ci aspettavamo, la classe appena sviluppata a partire da informazioni proposte come matrici 3 x 3 o comunque come tre vettori di dimensione tre giustapposti, effettua le operazioni tipiche di rotazione e traslazione. Si possono notare le rotazioni intorno ad un qualsiasi asse nella loro completezza anche come codice. Esse sono sviluppate facendo uso delle note matrici di rotazione, nel caso particolare il parametro a indica l'angolo di rotazione in radianti. Facilmente si possono realizzare le altre due rotazioni sotto forma di prototipi. Le trasformazioni realizzano le funzioni descritte nel box.

SISTEMI DI COORDINATE

Così come la terna di valori *(x,y,z)* di un vettore dipende dalla base che si sta usando; la terna *(x,y,z)* di un punto nello spazio a tre dimensioni dipende dal sistema di coordinate usato. Ad esempio, la punta della matita che adesso vedo sulla mia scrivania assume coordinate diverse a seconda se il sistema di riferimento è appunto la scrivania, la stanza, l'edificio, il centro della terra o quanto altro. Sostanzialmente ad una base bisogna associare un'origine. Si deve poter passare da un riferimento ad un altro. Rispetto alla struttura ereditaria di Fig. 1 implementiamo la classe *COORD_FRAME*

```
// Sistema di coordinate (base and origine)
class COORD_FRAME : public BASIS
{ public:
POINT O; //origine del sistema di coordinate, relative al
//riferimento padre (da non confondere la O con lo zero)
public:
// Costruttore vuoto
COORD_FRAME()
{}
// Costruttore con tre vettori e l'origine
COORD_FRAME(const POINT& o, const VECTOR& v0,
                      const VECTOR& v1,const VECTOR& v2)
: O ( o ),
BASIS ( v0, v1, v2 )
{}
COORD_FRAME(const POINT& o, const BASIS& b)
: O ( o ),
BASIS ( b )
{}
const POINT& position() const { return O; }
void position( const POINT& p ) { O = p; }
const POINT transformPointToLocal( const POINT& p )
const
{  //trasla all'origine del sistema di coordinata, si
                                    proietta nella base
  return transformVectorToLocal( p - O ); }
const POINT transformPointToParent( const POINT& p )
                                    const
{  //Trasforma le coordinate vettore e le trasla
                                    rispetto all'origine
return transformVectorToParent( p ) + O; }
//Trasla l'origine rispetto a un vettore dato
void translate( const VECTOR& v )
{ O += v; }};
```

Il metodo position restituisce la posizione. Gli altri si occupano di traslare e trasformare vettori rispetto all'origine. Le varie routine sono opportunamente commentate.

CORPI RIGIDI

L'ultimo passo è quello di produrre un oggetto corpo rigido, sulla base delle rigorose scelte fatte. Per chiarezza è opportuno scomporre ulteriormente in due classi *RIGID_BODY_STATE* che tiene traccia delle variazioni dinamiche del corpo. Viene derivato da *COORD_FRAME* e include la velocità lineare, la velocità angolare, il tensore di inerzia e il suo inverso. Passiamo al codice.

```
// Descrive lo stato dinamico del corpo rigido
class RIGID_BODY_STATE : public COORD_FRAME
{ public:
VECTOR V; //velocità lineare, metri/sec
VECTOR W; //velocità angolare, radianti/sec
MATRIX I; //tensore di inerzia con riferimento allo
                                    spazio terrestre,kg m m
MATRIX I_inv; //inversa del tensore di inerzia
public:
RIGID_BODY_STATE()
{}
RIGID_BODY_STATE(const VECTOR& v, const VECTOR& w)
: V (v),
```

- **CLASSICAL MECHANICS** ***T. L. Chow***
- **INTRODUCTION TO VECTOR ANALYSIS** ***A. D. Snider, H. F. Davis***
- **LINEAR ALGEBRA** ***S. Lang***
- **HIGH PERFORMANCE GAME PROGRAMMING IN C++** ***P. Perdiana*** **1988 (Computer game developer's conference)**
- **IL LINGUAGGIO C++** ***B. Stroustrup***

```
W (w)
{}
const VECTOR& velocity() const { return V; }
void velocity( const VECTOR& v ) { V = v; }
const VECTOR& angularVelocity() const { return W; }
void angularVelocity(const VECTOR& v) {W = v;}
const MATRIX& inertiaTensor() const { return I; }
const MATRIX& inverseInertiaTensor() const {
                                    return I_inv; }
//Calcola il tensore di inerzia e il suo inverso
//dal corrente vettore e il principale momento di inerzia
void calculateInertiaTensor( const VECTOR& ip );
};
```

NOTA

TRASFORMAZIONI

Per trasformare un vettore v dal suo spazio di riferimento (*body*) allo spazio terrestre moltiplichiamo le tre componenti *x, y* e *z* di *v* per i corrispondenti assi e addizioniamo tutti i risultati, come scritto di seguito:

VECTOR v_world = v.x* X_body + v.y*Y_body+v.z*Z_body

Per fare la trasformazione inversa dallo spazio terrestre a quello di riferimento del corpo, per ogni componente del corpo si deve fare il prodotto scalare tra il vettore e il riferimento; ecco come con riferimento ai metodi di *VECTOR*.

v_body.x = v.dot(X_body)
v_body.y = v.dot(Y_body)
v_body.z = v.dot(Z_body)

Per finire possiamo scoprire il sipario sull'oggetto ultimo anello della catena di derivazione. Oltre alla rotazione, traslazione, trasformazione, posizionamento, accelerazione lineare e angolare, per il corpo rigido possiamo rilevare le collisioni. Inoltre, abbiamo a disposizione uno stato precedente, utile nelle dinamiche di movimento di un corpo.

```
// Descrive un corpo rigido
class RIGID_BODY : public RIGID_BODY_STATE
{ public:
//stato dinamico precedente
RIGID_BODY_STATE PrevState;
//massa e inerzia rotazionale
SCALAR M; //mass, kg
VECTOR Ip; //principali momenti of inerzia, kg m m
public:
RIGID_BODY()
: M(1), Ip(1,1,1)
{}
RIGID_BODY( const SCALAR& m, const VECTOR& ip )
: M(m), Ip(ip)
{}
void mass( const SCALAR& m ) { M = m; }
SCALAR mass() const { return M; }
void inertiaMoments( const VECTOR& ip ) { Ip = ip; }
const VECTOR& inertiaMoments() const { return Ip; }
};
```

Ecco un esempio di codice C++ mediante il quale si simula il movimento di un corpo rigido sotto la forza e la torsione (sulla base delle equazioni di Newton ed Eulero). Come si può notare di seguito la stessa cosa in C è meno elegante. Ecco un'altra occasione per apprezzare la OOP.

```
// Soluzione C++
void RIGID_BODY::move( const SCALAR dt ) {
VECTOR F, T;
this->Translate( this->V * dt );
this->Rotate( this->W * dt );
ComputeForceAndTorque( F, T );
this->V += (1/this->M)* F * dt;
this->W += this->I_inv * (T - W.cross(I*W)) * dt;
}
// Codice equivalente C
void RIGID_BODY_Move( RIGID_BODY* pBody,
                                    SCALAR dt )
{
VECTOR F, T, t, v;
pBody->State.Frame.O.x += pBody->State.V.x * dt;
pBody->State.Frame.O.y += pBody->State.V.y * dt;
pBody->State.Frame.O.z += pBody->State.V.z * dt;
vmul( t, pBody->State.W, dt );//macro
RotateBasis( pBody->State.Frame.Basis, &t );
ComputeForceAndTorque( &F, &T );
s = 1/ pBody->M;
pBody->State.V.x += s * F.x * dt;
pBody->State.V.y += s * F.y * dt;
pBody->State.V.z += s * F.z * dt;
matrix_mul( t, pBody->State.I, pBody->State.W );
                                         //t = I*W
vcross( v, pBody->State.W, t ); //v = Wx(I*W)
vsub( t, T, v ); //t = T - Wx(I*W)
matrix_mul( v, pBody->State.I_inv, t ); //v = I_inv*(
                                      T - Wx(I*W))
vmul( t, v, dt ); //t = I_inv * (T - Wx(I*W)) * dt
pBody->State.W.x += t.x;
pBody->State.W.y += t.y;
pBody->State.W.z += t.z;
}
```

CONCLUSIONI

Abbiamo visto come un buon insieme di dati strutturati associa ad un corretto e chiaro progetto OOP garantisca risultati di semplice comprensione ed efficienti. Si tratta solo di riservare inizialmente un po' di tempo alla progettazione.
Concludo questa esperienza soddisfatto di aver costruito del codice semplice ed efficiente, e sperando che possa essere da spunto per chi ha intenzione di sviluppare in tale ambito.
Nel ricercare nuovi ed interessanti argomenti, porgo un saluto a tutti i lettori.

Fabio Grimaldi

Quesiti elementari dai sorprendenti spunti algoritmici

Ritorno al classico

Le torri di Hanoi è un classico rompicapo che ha appassionato gli amanti del genere per molti anni. L'avvento del computer ha dato alla questione nuova linfa e tuttora è riconosciuto come uno dei più interessanti esercizi.

Tra un rompicapo e l'altro di tanto in tanto spuntano problemi classici che mantengono intatto il loro fascino e la forte valenza didattica. È il caso delle torri di Hanoi che oltre ad essere un gradevolissimo gioco da fare anche disconoscendo l'esistenza del mostro al silicio, ha importanti riferimenti alle soluzioni di algoritmi ricorsivi. Senza entrare nei particolari della risoluzione che lasciamo di consueto al lettore, si sappia che quello delle torri di Hanoi è senza dubbio l'esempio più fulgido che viene proposto per esplicare la ricorsione. Ma facciamo un passo indietro. I lettori dello scorso appuntamento reclamano attenzione.
Abbiamo, infatti seminato, nel terreno della nostra capacità risolutiva, due nuovi problemi, per la verità non molto difficoltosi ma dagli spunti intriganti.
Affronteremo nei prossimi due paragrafi la risoluzione ai problemi proposti, ossia il gioco di Euclide e il trucco magico di Bechelet. Per terminare, formuleremo l'enigma delle citate torri di Hanoi.

IL GIOCO DI EUCLIDE

Il gioco proposto è, nella sua formulazione, banale, ma come vedremo propone utili spunti di riflessione nella fase di soluzione. Dopo aver sorteggiato due numeri bisogna a turno scrivere su un foglio un numero che sia la differenza di due presenti. Ovviamente, al primo passo ci sarà una sola possibilità che è la differenza dei due numeri iniziali. Perde chi non ha più alcun numero da scrivere. La cosa sorprendente è che si può stabilire subito quanti siano i numeri che si possono produrre con la sola conoscenza del massimo comun divisore *(mcd)* tra i due.
Avremo quindi la possibilità di rivisitare l'algoritmo per il calcolo del mcd proposto da Euclide, un altro classico!
Si parte dalla considerazione che la differenza tra due qualsiasi numeri è divisibile per mcd degli stessi. Supponendo che i due numeri *N* e *M* siano tali che *N>M* (trascurando il caso limite in cui siano uguali) allora i soli numeri che si possono ottenere per differenza, così come formulato il gioco, sono multipli di *mcd(N,M)* (massimo comun divisore di *N* e *M*). Si faccia qualche esempio a riprova di quanto scritto. Cosicché è possibile individuare il numero di interi che si possono scrivere, essi sono pari a *N/mcd(N,M),* compresi in tale quantità anche gli iniziali *N* e *M.* La conoscenza del numero di mosse che porta all'esaurimento del gioco è un'informazione fondamentale per l'esito finale del gioco. Così se l'avversario non sa il trucco, potremo scegliere se iniziare per primi o meno a seconda che il numero di mosse sia dispari o pari, all'ovvio fine di vincere la partita.
Facciamo qualche esempio. Se i due numeri sono 10 e 6, allora *mcd(10,6)* è 2 da cui deduciamo il numero di mosse *(10/2=5)* pari a cinque. Essendo dispari vinceremo se avremo a disposizione la prima mossa. Un altro esempio: siano i due numeri 36 e 9 il mcd(36,9) = 9 da cui si deduce *N/mcd(N,M)* pari a *36/9=4*. Questa volta per vincere la partita, il numero pari ci impone di cominciare per secondi.
Riesaminiamo con piacere l'algoritmo di Euclide per l'individuazione del massimo comun divisore tra due numeri. Solitamente è tra i primi che si apprendono quando si affronta la programmazione e la costruzione di algoritmi. Siano *a* e *b* due numeri di cui si vuole calcolare *mcd(a,b),* supposto che *a>b,* il metodo è basato su due considerazioni. La prima indica che se *b* è divisibile per *a* allora, ovviamente, *mdc(a,b) =b*, ricordando che il più grande divisore di un qualsiasi numero è se stesso. Esempio, *mcd(14,7)=7.* Si rammenta, inoltre, che l'insieme di riferimento sono i numeri naturali, non si tiene conto quindi, di numeri negativi.
La seconda considerazione che esprime la vera natura dell'algoritmo è basata sull'espressione *a = b*t + r,* con *t* e *r* numeri naturali (interi non negativi). Se tale espressione è valida allora *mcd(a,b) = mcd (b,r).* Infatti ogni comune divisore di a e b è anche divisore di *r.* Così *mcd(a,b)* divide *r.*
Ma naturalmente *mcd(a,b)* e divisibile per *b.* Perciò *mcd(a,b)* è divisibile per *b* quanto per *r,* dunque *mcd (a,b) <= mcd (b,r).* L'inverso è anche vero poiché ogni divisore di *b* e *r* lo è anche di *a.* Ecco un esempio.

RICORSIONE

Un definizione è ricorsiva se è descritta mediante se stessa. I numeri naturali o la potenza di un numero possono essere spiegati matematicamente in modo ricorsivo. Vediamo come. L'insieme dei numeri naturali è definito secondo le seguenti due regole:

1) 1 è un numero naturale
2) Il successore di un numero naturale è ancora un numero naturale.

Da questa definizione per il primo punto risulta che il numero 1 è un numero naturale. La seconda regola applicata la prima volta indica che il numero 2 è un numero naturale visto che è il successore di un altro numero naturale, ovvero, 1. Applicando nuovamente la seconda regola ci si rende conto che anche il numero 3 appartiene all'insieme dei numeri naturali. Continuando ad applicare un numero infinito di volte tale regola si nota che i numeri 4,5,6,7, .. appartengono tutti ai numeri naturali. L'insieme è costituito da infiniti elementi.

Sia *a = 2322, b = 654.*

2322 = 654*3 + 360	mcd(2322, 654) = mcd(654, 360)
654 = 360*1 + 294	mcd(654, 360) = mcd(360, 294)
360 = 294*1 + 66	mcd(360, 294) = mcd(294, 66)
294 = 66*4 + 30	mcd(294, 66) = mcd(66, 30)
66 = 30*2 + 6	mcd(66, 30) = mcd(30, 6)
30 = 6*5	mcd(30, 6) = 6

perciò *gcd(2322,654)=6*

IL TRUCCO MAGICO DI BECHELET

Esaminiamo il secondo enigma proposto nello scorso appuntamento. Dopo aver scelto una coppia di numeri tra *N*(N+1)/2,* si devono indovinare gli stessi presentati singolarmente in forma di matrice e con la sola indicazione della loro collocazione come riga all'interno della matrice (dimensione *N x (N+1)*). Il calcolo combinatorio, che anche in passato ci ha spesso aiutato, risulterà prezioso in questa occasione. La formulazione da adottare per la ricerca della soluzione prevede il minor numero di oggetti che si possono combinare in tutti i modi possibili. Se *N=4* abbiamo *N*(N+1)/2* combinazioni, ecco quali: *{1, 1}, {1, 2}, {1, 3}, {1, 4}, {2, 2}, {2, 3}, {2, 4}, {3, 3}, {3, 4}* e *{4,4}*. Gli oggetti vengono presentati in forma matriciale *5 x 4 (N+1) x N* come mostrato di seguito:

{1, 1}, {1, 1}, {1, 2}, {1, 3}, {1, 4},
{2, 1}, {2, 2},{2, 2}, {2, 3}, {2, 4},
{3, 1}, {3, 2}, {3, 3}, {3, 3}, {3, 4},
{4, 1}, {4, 2}, {4, 3}, {4, 4}, {4, 4},

Si noterà che si tratta di una matrice speculare e anche se in ordine diverso le coppie di oggetti si presentano due volte. Nel caso di oggetti identici vi saranno semplicemente due volte le stesse coppie. Si distinguono quindi due triangoli uno inferiore ed uno superiore in cui sono presenti tali coppie che definiremo simili. Ed è proprio questa duplicazione la chiave del trucco di Bechelet, poiché se ad ognuna delle 10 coppie di *N* elementi *(4)* viene associata una coppia di numeri da *1* a *20;* esempio *(1,2), (3,4), (5,6), (7,8), (9,10), (11,12), (13,14), (15,16), (17,18)* e *(19,20);* si innesta la soluzione al gioco. Ora, poiché ogni coppia nella matrice esaminata risulta due volte (anche se gli elementi sono proposti in ordine diverso), si potrà associare ogni numero ad un elemento della matrice. Operazione, questa ultima da svolgere con accortezza: se ad esempio si associa il numero 5 al paio *{1, 3}* ovviamente il suo accoppiato 6 corrisponderà al reciproco *{ 3, 1}*.

1,	*2,*	*3,*	*5,*	*7*
4,	*9,*	*10,*	*11,*	*13*
6,	*12,*	*15,*	*16,*	*17*
8,	*14,*	*18,*	*19,*	*20*

Così, basta tenere traccia della corrispondenza numero-coppia per svelare il mistero. Infatti, se la coppia pensata è *5, 6,* nel consultare la matrice si indicheranno le coppie *1* e *3* corrispondenti ai numeri *5* e *6*. Ecco rilevato l'enigma. Per concludere ricordo che i *N*(N+1)* numeri (nell'esempio *20*) possono essere prodotti casualmente, è necessario però fare l'associazione tra i numeri e le coppie della matrice di base.

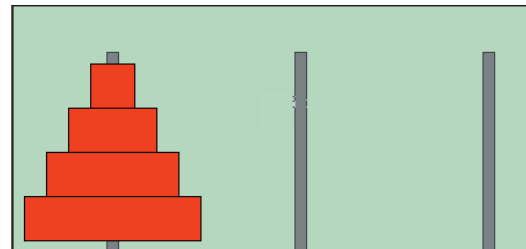



Fig. 1: **Posizione iniziale del gioco Torri di Hanoi.**

TORRI DI HANOI

È arrivato il momento di proporre un nuovo enigma. Come scrivevo si tratta di un problema classico. Si hanno ha disposizione tre paletti e *n* dischi di dimensioni diverse. Inizialmente i dischi sono impilati nel primo dei paletti in modo da formare una torre (il più grande alla base e sopra i dischi di dimensione sempre più piccoli, quello in cima sarà ovviamente il più piccolo). Si vogliono portare tutti i dischi dal paletto *1* al paletto *3* in modo che formino una torre come nel caso iniziale. Devono però essere rispettate le seguenti regole:

1) ad ogni passo un solo disco viene spostato;

2) un disco non può essere posto su un disco più piccolo;

Il paletto *2* può essere utilizzato come sosta temporanea per i dischi. Premesso che la soluzione solo manuale, ossia senza l'uso del computer, è già molto interessante, è di notevole importanza la risoluzione algoritmica. Per questa hanno un aspetto fondamentale l'implementazione ricorsiva e l'analisi delle tracce della variazione dello stack di sistema.

CONCLUSIONI

Rivolgere l'attenzione a problemi classici oltre che un utile esercizio di ripasso, è essenziale come atto di formazione rispetto ad alcune tipologie di problemi. E poi, non so voi, ma su di me, tali problemi esercitano sempre un notevole fascino. Ad esempio, chi l'avrebbe detto che Euclide oltre ad essere un grande studioso era un giocherellone!
La prossima volta, dopo questo ritorno al classico, questa piacevole parentesi, saranno di scena nuovi enigmi. Vi aspetto!

Fabio Grimaldi

Python Italia

Di linguaggi di programmazione ne esistono tanti. Alcuni, sono universalmente conosciuti ed applicati, altri, restano destinati ad una nicchia oppure circoscritti a particolari ambiti applicativi.

Python è il nome di un linguaggio relativamente giovane, che da strumento di nicchia spinge sempre più per diventare un caposaldo dell'informatica moderna. Le carte in regola per il salto, in effetti, le ha tutte.

COS'È PYTHON?

Python è stato ideato ad Amsterdam da Guido Van Rossum (*http://www.python.org/~guido/*), a cavallo fra gli anni '80 e gli anni '90. Tecnicamente parlando Python è stato concepito come un linguaggio di scripting per ambienti UNIX, per estendere e semplificare la scrittura degli script destinati alla shell, più di quanto già non facesse Perl con la sua sintassi a tratti oscura (da qualche parte ho letto, a proposito della sintassi di Perl, "stile passeggiata del micio sulla tastiera"). Python, rispetto ad altri linguaggi analoghi, è orientato agli oggetti, ed abbina alla semplicità di un linguaggio di scripting la completezza e la potenza dei più tradizionali linguaggi di programmazione.

E JAVA: COMPILATO O INTERPRETATO

Python, proprio come Perl e Java, è sia interpretato sia compilato. Il compilatore di Python traduce il codice sorgente in una codifica intermedia, un bytecode, che una macchina virtuale può poi interpretare ed eseguire con prestazioni migliori rispetto a quelle ottenibili con un linguaggio solamente interpretato. L'interprete di Python, necessario per eseguire qualsiasi codice scritto con questo linguaggio, rappresenta dunque tale macchina virtuale. Come avviene con Perl, è possibile mandare in esecuzione un sorgente Python, che l'ambiente del linguaggio trasformerà al volo in bytecode, prima dell'esecuzione vera e propria. In maniera simile a Java, il bytecode ottenuto alla prima esecuzione del sorgente viene conservato su disco, in modo che agli avvii successivi non sia più necessaria la compilazione in linguaggio intermedio (salvo modifiche del sorgente, ovviamente, ed in questo Python ricorda anche la tecnologia JSP).

Fig. 1: La homepage del Sito.

Python, Perl e Java, dunque, hanno tutti e tre bisogno di un ambiente di esecuzione. Un'architettura di questo tipo, è evidente, favorisce la portabilità del codice, purché la particolare piattaforma hardware e software puntata disponga di un ambiente di esecuzione idoneo. Python, da questo punto di vista, è ben messo. Esistono interpreti per tutte le maggiori piattaforme in circolazione, e non solo: UNIX, Linux, Windows, Mac OS, Amiga, BeOS, OS/2 e così via. Inoltre, esiste una particolare implementazione dell'ambiente di Python, chiamata Jython e realizzata in Java. Jython compila un sorgente Python in bytecode Java, che può essere eseguito all'interno di qualsiasi macchina virtuale Java-compatibile. Laddove è disponibile Java è automaticamente disponibile anche Python. Riguardo l'attuale diffusione dell'interprete di Python, c'è da segnalare che le principali distribuzioni Linux lo includono in maniera predefinita, giacché il linguaggio si è ritagliato un'importante fetta applicativa in ambiente GNU/Linux. Gli utenti Windows, invece, possono tranquillamente scaricare ed installare per proprio conto l'ambiente, che ha un peso tutto sommato modesto.

I PRO DI PYTHON

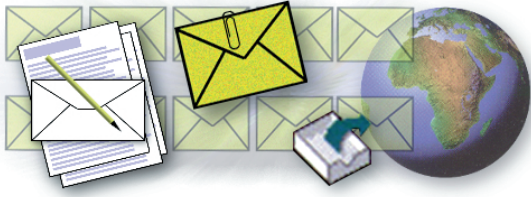
Molti sono gli aspetti positivi di Python. Anzitutto è Software Libero. Poi è portabile, ed è anche efficiente, grazie al trucco della compilazione in bytecode. Python ha una sintassi chiara. Croce e delizia del linguaggio è rappresentata dall'indentazione, che in Python assume significato rilevante ai fini della programmazione. L'indentazione, in altri linguaggi, è usata solo per mantenere ordinato il codice, ed è lasciata a discrezione del programmatore. In Python, al contrario, l'indentazione sostituisce i blocchi delimitati da parentesi per la strutturazione del codice. Python, in parole povere, costringe il programmatore a mantenere ordinati i propri sorgenti. Alcuni puristi ritengono la scelta un grado di libertà in meno, altri sviluppatori trovano l'idea semplicemente noiosa, ma molti altri ne apprezzano i vantaggi.

NELLA PROGRAMMAZIONE MODERNA

Python piace molto agli sviluppatori, soprattutto a quelli degli ambienti del Software Libero e dell'Open Source. Facendo un giro su SourceForge.net è possibile verificare l'affermazione. Se questo non bastasse, vi elenco alcuni nomi famosi che fanno uso di Python: Google, Yahoo, IBM, Red Hat (Anaconda, il loro tool di installazione e configurazione del sistema operativo, è realizzato in buona parte con Python), NASA e Industrial Light & Magic (quelli degli effetti speciali di Star Wars, per intenderci).

PYTHON ITALIA

Il principale sito di riferimento per Python, al quale fa capo tutto il mondo, è *http://www.python.org/*, naturalmente in lingua inglese. L'Italia è rimasta inizialmente insensibile al fascino di Python, fin quando dalla fine del secolo scorso è andata a formarsi la prima comunità di sviluppatori amanti di questo linguaggio. Il risultato di questo incontro è dato dal sito Python Italia (*http://www.python.it/*) e dal newsgroup it.comp.lang.python. Il sito, in particolar modo, rappresenta il principale punto di riferimento per gli sviluppatori Python di casa nostra. Da Python Italia è possibile arrivare velocemente al download degli interpreti del linguaggio e di tutti gli strumenti collegati. Il sito ospita inoltre guide, riferimenti, FAQ e tutorial in lingua italiana. Alcune sezioni risultano ancora incomplete ed in lavorazione. L'augurio è che la comunità possa crescere ed espandersi rapidamente, per dare a Python lo spazio e la visibilità che indubbiamente il linguaggio merita anche in Italia.

Carlo Pelliccia
carlo.pelliccia@ioprogrammo.it

INBox L'esperto risponde...

Pulsanti grassi in C++

Ciao ragazzi, volevo sapere se è possibile, e in caso come si fa, a mettere il testo bold sui pulsanti! Ciao e Grazie!

All'interno della *OnInitDialog* della tua finestra, aggiungi il seguente codice:

```
BOOL CDialogo::OnInitDialog()
{ CDialog::OnInitDialog();
// TODO: Add extra initialization here
CFont * mFont;
mFont = new CFont;
mFont->CreateFont(/*int nHeight*/ 14,
    /*int nWidth*/ 0, /*int nEscapement*/ 0,
/*int nOrientation*/ 0, /*int nWeight*/
    FW_EXTRABOLD, /*BYTE bItalic*/ 0, /*
                        BYTE bUnderline*/ 0,
/*BYTE cStrikeOut*/ 0, /*BYTE nCharSet*/
                        DEFAULT_CHARSET,
/*BYTE nOutPrecision*/ OUT_CHARACTER_PRECIS,
                  /* BYTE nClipPrecision */
                  CLIP_CHARACTER_PRECIS,
/*BYTE nQuality*/ DEFAULT_QUALITY,
                  /*BYTE nPitchAndFamily*/
              DEFAULT_PITCH | FF_DONTCARE,
/*LPCTSTR lpszFacename*/ _T("Arial"));
CWnd * pwnd = this->GetDlgItem(IDOK);
  // ID del pulsante di cui vuoi modificare il font
pwnd->SetFont(mFont,TRUE);
return TRUE; // return TRUE unless you set
                       the focus to a control
// EXCEPTION: OCX Property Pages should
                              return FALSE }
```

Per modificare i parametri del font, studiati il metodo *CreateFont* della classe *CFont*...

Rappresentazioni isometriche con DirectX

Ho letto con attenzione il primo articolo su DirectX comparso nel numero di Febbraio di ioProgrammo. Grazie agli articoli ed ai tutorial ho potuto testare facilmente le funzionalità di base di DirectX. Ho riscontrato subito un problema: è possibile ottenere una rappresentazione isometrica, quindi non prospettica? Ponendo la camera molto distante e un angolo di vista molto stretto si raggiunge una approssimazione della vista isometrica, ma ho paura che poi si perda nella precisione dei calcoli perché la matrice di proiezione viene malcondizionata. Ha qualche suggerimento da fornirmi?

Paolo Braschi

Risponde Carlo Zoffoli

Hai assolutamente ragione, infatti non si usa Direct3D per i giochi isometrici, ma DirectDraw (che non è assolutamente una libreria sorpassata). Fra l'altro, usare Direct3D nelle isometrie è un enorme spreco di risorse: con DirectDraw puoi caricare quattro bitmap per ogni, ad esempio, personaggio che vuoi rappresentare e scegli quale caricare a seconda se il personaggio in questione "punta" verso l'alto, il basso, a destra o a sinistra. E così per ogni altro oggetto in movimento nel gioco isometrico. Permettimi di consigliarti "**Isometric Game Programming with DirectX 7.0 w/CD**" di *Ernest Pazera, et al* (Paperback), che puoi trovare su *Amazon.Com*. Anche se parla di DirectX 7 e di C++ è comunque un buon riferimento (DirectDraw non è quasi per niente cambiato dalla versione 7, e le chiamate C++ sono molto simili a quelle C#). Comunque se continui a seguire gli articoli su DirectX in .NET di ioProgrammo vedrai che verrà affrontato anche questo argomento, più in là.

I "profani" di J2EE

Scusate la banalità della domanda, ma non mi sono ancora buttato nel mondo Enterprise di Java: che significa esattamente il termine "deploy" ?

Matteo Difuria

Risponde Krystal

La fase di "deploy" altro non fà che "installare" la web application creata sul particolare Application Server utilizzato. In genere "deployare" assume un senso concreto in presenza di EJB (Enterprise Java Beans), che non vengono solo "copiati" nella cartella giusta del proprio Application Server, ma anche "configurati" (in genere tramite file XML), al fine di essere resi operativi e utilizzati correttamente dall'Application Server (e purtroppo ogni Application Server ha la sua semantica e i suoi tool di configurazione).

Processi figli e demoni

Salve a tutti! Vi espongo il mio problema: in C ho fatto un programma che genera dei processi figli e comunica con loro tramite pipe e socket. il programma in questione crea anche un processo che resterà poi orfano diventando demone. La mia domanda ora è questa: come gestire creazione cancellazione etc... dei processi con Java? Si puo fare vero? Grazie a tutti!!

Risponde Krystal

Java supporta unicamente Thread. Come è facile immaginarsi, un Thread è anch'esso un oggetto e nel particolare. Per "creare" un Thread hai 2 alternative:

a) implementare l'interfaccia *java.lang.Runnable* (e di conseguenza il metodo *"run"* che contiene il codice da eseguire..) e passare tale classe come argomento ad uno dei costruttori di *Thread (Thread (Runnable target)*, ad esempio)

```
class TuaClasse implements Runnable
{ ...
public void run()
{ ...}}
//successivamente (nel main ad esempio)
TuaClasse tua = new TuaClasse();
new Thread(tua).start();
```

b) estendere direttamente *java.lang.Thread* ridefinendo opportunamente il metodo *"run"* (in effetti Thread implementa a sua volta *Runnable...*)

- per "eseguire" un *Thread* basta richiamare il metodo *"start"*

```
class TuaClasse extends Thread
{ ...
public void run()
{ ...}
}
//successivamente (nel main ad esempio)
TuaClasse tua = new TuaClasse();
tua.start();
```

- per "interrompere" un *Thread* basta richiamare su di esso il metodo *interrupt();*
- per "tante altre cose" ti consiglio vivamente di dare uno sguardo alla documentazione ufficiale che trovi anche al link *http://java.sun.com/j2se/1.5.0/docs/ api/*, cercando *"Thread"* tra *"All Classes"*.

P.S. Potrebbe esserti utile il metodo *setDaemon* e, se sei interessato a gestire contemporaneamente + *Thread*, dai uno sguardo anche a *"ThreadGroup"* (per cambiare la priorità ad un gruppo di *Thread*, ad esempio).

VB.NET, setup di un'applicazione

Ciao a tutti! Ho la necessità di distribuire la mia applicazione salvata su un CD a *n* persone; sui PC di queste *n* persone non è installato vb.net nè .net framework. Lanciando il setup di installazione dell'applicazione viene richiesta anche l'installazione di .net framework. Mi piacerebbe evitare alle persone di dover seguire una sequenza di passi per l'installazione facendo loro lanciare solo l'eseguibile del mio setup.
Qualcuno sa dirmi come includere .net framework nel setup di installazione di un'applicazione? Ringrazio anticipatamente, ciao.

Dalù

Risponde Fabio Cozzolino

Devi utilizzare il bootstrapper .NET. Puoi trovare il plug-in e le instruzioni a questo indirizzo: *http://msdn.microsoft.com /vstudio/downlo...s/bootstrapper/*

Button in VB.NET

In VB6 i *commandbutton* prevedevano la proprieta' *tooltiptext.* In VB.NET tra le proprietà dei button non ho trovato niente di simile, non esiste qualcosa di simile? Ciao e grazie

Risponde Fabio Cozzolino

In VB.Net questa funzionalità è stata sostituita con il controllo *ToolTip*. Questo permette un maggiore controllo oltre alla possibilità di usufruire di uno strumento molto più potente. Devi quindi aggiungere un controllo *ToolTip* al tuo form e collegarlo ai vari controlli da implementare. Questo è un esempio tratto da MSDN:

```
Private Sub Form1_Load(sender As Object, e
    As System.EventArgs) Handles MyBase.Load
' Create the ToolTip and associate with the
                               Form container.
Dim toolTip1 As New ToolTip()
' Set up the delays for the ToolTip.
toolTip1.AutoPopDelay = 5000
toolTip1.InitialDelay = 1000
toolTip1.ReshowDelay = 500
' Force the ToolTip text to be displayed
              whether or not the form is active.
toolTip1.ShowAlways = True
' Set up the ToolTip text for the Button and
                                     Checkbox.
toolTip1.SetToolTip(Me.button1, "My button1")
toolTip1.SetToolTip(Me.checkBox1, "My
                                   checkBox1")
End Sub
```

ovviamente sulla documentazione del framework trovi una marea di informazioni.

Esistenza file in C#

Salve a tutti, come faccio a verificare se esiste un determinato file nella directory principale dell'applicazione?!? In vb.net uso dir qual'è l'equivalente in c#? Grazie

Beziel

Risponde Antonio Pelleriti

Puoi utilizzare il metodo statico Exists della classe File, ad esempio:

```
if (File.Exists(percorso))
{
  //il file esiste
}
```

Microsoft Agent

Ho trovato molto interessante l'articolo sulla rivista riguardante Agent, ed ho subito fatto qualche prova, ma la voce dei personaggi è sempre la stessa e ciò rende la cosa meno friendly, dunque mi kiedevo se fosse possibile cambiare la voce dei personaggi.

Master

risponde Roberto Allegra

Ciao, ti ringrazio per l'apprezzamento. Purtroppo per motivi di spazio non è stato possibile pubblicare un laterale che riguardava la questione voci/engines.
Per trattarla bene ci vorrebbe molto spazio, e molto tempo (un articolo a parte). Agent si basa sui TTS quindi, ottenendo l'identificativo del motore (tramite *Speech API* o controllo *Text-To-Speech*), puoi cambiare il set per la voce, usandone un modeID precompilato. Per le *L&H* i valori predefiniti sono i seguenti:

```
Barbara, Italian, L&H TTS3000 {
  7EF71700-A92D-11d1-B17B-0020AFED142E}
Stefano, Italian, L&H TTS3000 {
  7EF71701-A92D-11d1-B17B-0020AFED142E}
```

Puoi inoltre cambiare i valori di *Pitch* e *Speed*, anche se non direttamente da controllo.
Per cambiare i valori di un personaggio già esistente "in corsa" puoi usare i tag:

```
"\spd=180\Ciao!", ad esempio, imposta la
     velocità 180 per la stringa in questione.
```

Puoi comodamente impostare per il personaggio una stringa default corrispondente ad un insieme di tag che definiscono il personaggio, ad esempio:

```
VoceMerlin = "\spd=180\\pit=150\"
```

e poi richiamarla quando ti serve, associandola al metodo speak

```
Merlin.Speak VoceMerlin & "Ciao!"
```

Questo è un modo molto comodo di risolvere la questione. Nel caso in cui sia tu a creare un personaggio nuovo tramite ACE (prossimo articolo) puoi decidere i valori da associare per default a velocità e pitch. Ti consiglio di scaricare l'SDK di Agent, c'è una buona sezione sull'argomento.
Ciao

PER CONTATTARCI
e-mail: ioprogrammo@edmaster.it
Posta: Edizioni Master,
Via Cesare Correnti, 1 - 20123 Milano

ON LINE

WEB DEVELOPER

Un sito dedicato a tutti gli sviluppatori web e a chi vuole diventarlo; una vasta sezione dedicata a CSS, DHTML, HTML, .NET, XML, ASP, PERL, JavaScript, PHP.

www.webdeveloper.com

PYTHON

Python è stata una parte importante in Google sin dall'inizio, e rimane tale mentre il sistema cresce e si evolve. Oggi dozzine di ingegneri di Google usano Python, e stiamo cercando sempre più gente brava in questo linguaggio." dice Peter Norvig, direttore della qualità di ricerca al Google center.

www.python.it

ACKY

Tutorial, archivi, forum per JavaScript, Visual Basic, CGI Scripts, PhotoShop, Flash, HTML, Visual Basic, Perl

www.acky.net

Biblioteca

FLASHMX 2004 ACTIONSCRIPT

Il corso ufficiale per i principianti e intermedi di Flash MX 2004 ActionScript. Lezioni passo passo, progetti concreti e spiegazioni complete per percepire la logica che sta dietro gli script e per capire come, coordinati tra loro, completino un progetto. Chi conosce già ActionScript 1.0, si accorgerà che la versione 2.0 non è molto diversa, salvo alcune piccole ma essenziali differenze. Tra gli argomenti trattati:

- La nuova sintassi di ActionScript 2.0.
- Classi di oggetti predefinite (Color, Sound e Array) e classi personalizzate
- Utilizzazione ed elaborazione dei dati nei progetti.
- Progettazione con la logica condizionale.
- Creazione di un'interfaccia interattiva.
- Creazione di un'applicazione chat a più stanze usando un server socket XML.
- Stampa del contenuto Flash in modo interattivo.

Il CD-ROM allegato contiene i file necessari per completare i progetti e verificarli con i risultati ottenuti.

Difficoltà: Media • Autori: Derek Franklin - Jobe Makar • Editore: Mondadori Informatica http://education.mondadori.it • ISBN: 88-04-53127-4 • Anno di pubblicazione: 2004 • Lingua: Italiano Pagine: 784 • Prezzo: € 50.00

INTERMEDIATE ROBOT BUILDING

Per gli amanti della robotica APress propone il seguito di "Robot Building for Beginners". Il testo offre un apprendimento "real-world", in condizioni reali, che solo un veterano costruttore di Robot può offrire. Chi mastica l'inglese non avrà problemi a capire le tecniche giuste per creare robot esploratori o combattenti capaci di evitare ogni ostacolo, senza rischiare di far esplodere i condensatori! Le istruzioni passo passo, gli schemi dettagliati dei circuiti, una gran quantità di foto descrittive, i meccanismi più ingegnosi vi permetteranno di evitare ogni errore. Lo stile di scrittura è estremamente semplice da capire, in poche mosse diventerete provetti "Robot Builder".

Difficoltà: Media • Autore: David Cook • Editore: Apress **www.apress.com • ISBN: 1-59059-373-1 Anno di pubblicazione: 2004 • Lingua: Inglese • Pagine: 442 • Prezzo: $ 34.99**

L'ARTE DELL'HACKING

Il primo vero manuale che spiega le tecniche di hacking più diffuse, senza tralasciare alcun dettaglio tecnico. Partendo dal presupposto che conoscere i metodi di attacco possa rappresentare un'arma in più di difesa, Jon Erickson guida il lettore esperto in un vero e proprio percorso di iniziazione alle tecniche hacker. Scritto in un modo chiaro, conciso e organizzato, questo libro vi spiegherà come:

- Sfruttare buffer overflow e format string per creare gli exploit
- Scrivere shellcode polimorfico stampabile in ASCII
- Sconfiggere i meccanismi di protezione dello stack tramite il ritorno alla libc
- Redirezionare il traffico di rete, nascondere le porte aperte e dirottare le connessioni TCP
- Decifrare il traffico criptato su reti wireless 802.11b mediante l'attacco FMS

Difficoltà: Medio-Alta • Autore: Jon Erickson • Editore: Apogeo www.apogeonline.com ISBN: 88-503-2280-1 • Anno di pubblicazione: 2004 • Lingua: Italiano • Pagine: 256 Prezzo: € 24.00